



NVIDIA DRIVE OS Linux SDK Developer Guide

Chapter 1. Overview

The NVIDIA DRIVE® OS 6.0 Linux Software Development Kit enables application development with NVIDIA DRIVE Orin™ SoC for Automotive. At a high level, the SDK consists of:

- > DRIVE OS foundation components including bootloaders, a Type 1 hypervisor, and virtualization.
- > GuestOS virtual machines including Linux operating systems for system software and application development.

The DRIVE OS Software Development Kit (SDK) provides industry standard APIs including NvMedia, CUDA and TensorRT. The CUDA and TensorRT modules included in DRIVE OS 6.0 software releases are compatible only with the Automotive DRIVE AGX Orin Platform. As such, they must only be used with DRIVE OS. These modules must not be used standalone as they are not compatible with other NVIDIA devices.

What is NVIDIA DRIVE OS 6.0?

NVIDIA DRIVE® OS 6.0 is the reference operating system and associated software stack designed specifically for developing and deploying autonomous vehicle applications on DRIVE Orin-based hardware. DRIVE OS 6.0 delivers a safe and secure execution environment for safety-critical applications, providing services such as secure boot, security services, firewall, and over-the-air updates.

The included foundational software stack consists of NVIDIA® CUDA® libraries, NVIDIA TensorRT™, NvMedia, and other components optimized to provide direct access to DRIVE AGX Orin hardware acceleration engines.

What is NVIDIA DRIVE OS 6.0 SDK?

DRIVE OS 6.0 SDK consists of DRIVE Orin software, libraries, and tools to build, debug, profile, and deploy applications for autonomous vehicles and self-driving cars across the CPU, GPU and other DRIVE AGX Orin hardware acceleration engines. These development tools provide optimized workflows for parallel computing and deep learning development.

In order to maximize productivity, DRIVE OS 6.0 SDK leverages industry standard tools, technologies, and APIs to provide a familiar and comfortable high-productivity development environment.

Getting Started with NVIDIA DRIVE OS SDK

Topic	Where to Learn More
Installation	See Installation for information on installing or upgrading a DRIVE OS SDK installation on a host machine.
Flashing	See Flashing DRIVE AGX Orin for information on updating the system image and associated firmware on a DRIVE AGX Orin system.
Architecture	See DRIVE OS Architectural Overview for an architectural overview of the DRIVE OS stack.

What is NVIDIA DRIVE OS 6.0 PDK?

NVIDIA DRIVE® OS 6.0 Platform Development Kit (PDK) is used to modify DRIVE OS 6.0 to run on non-reference DRIVE AGX Orin-based hardware platforms. The NVIDIA DRIVE OS PDK requires specific agreements with NVIDIA. Consult with your NVIDIA Customer Support Engineer for more information.

Additional Platform Components

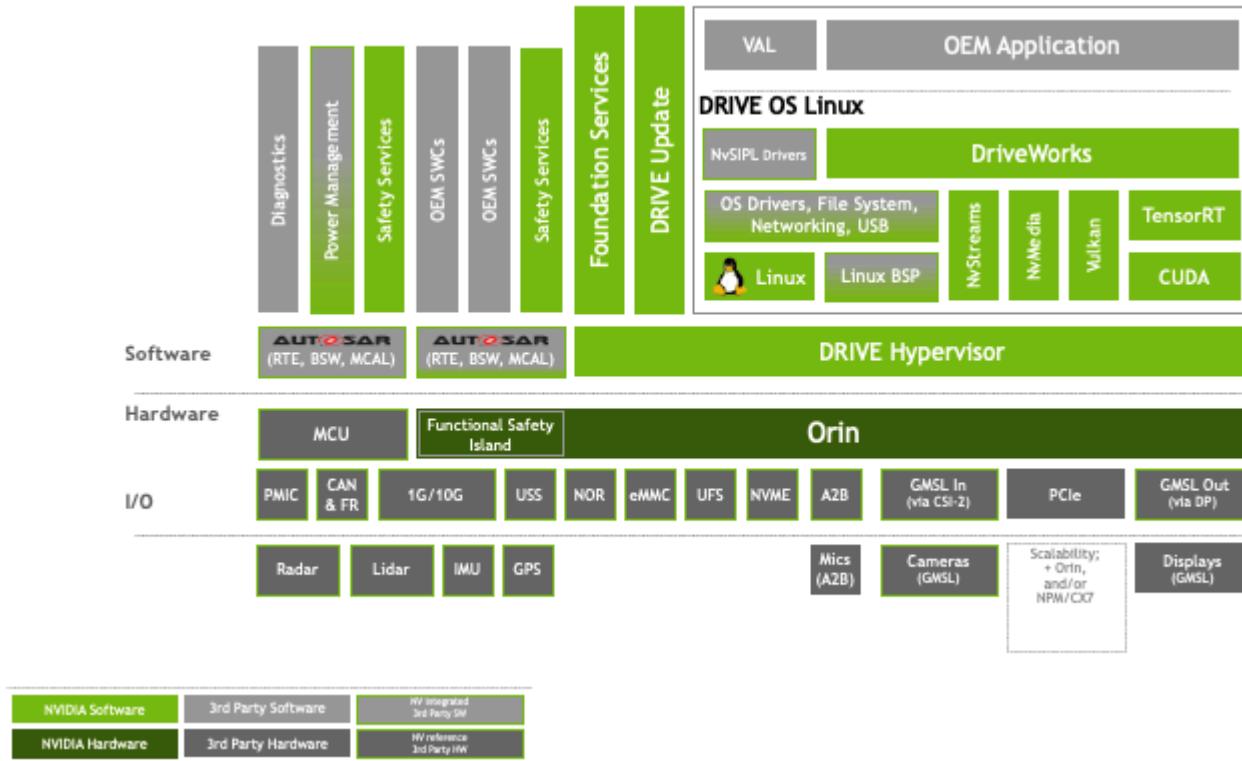
In addition to the Foundation services components and the DRIVE OS-specific components, additional components are available. These components are provided separately for customizing the platform development and include:

Components	Description
CUDA	<p>CUDA® is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs.</p> <p>Consult the CUDA Samples provided as an educational resource.</p> <p>Consult the CUDA Computing Platform Development Guide for general purpose computing development.</p>
cuDNN	<p>The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. cuDNN is part of the NVIDIA Deep Learning SDK.</p> <p>Consult the cuDNN Deep Neural Network Library of primitives for deep neural network development.</p>
TensorRT	<p>NVIDIA TensorRT™ is a high-performance deep learning inference optimizer and runtime that delivers low latency, high-throughput inference for deep learning applications.</p> <p>Consult the TensorRT Documentation for deep learning development.</p>

1.1 Platform Software Stacks

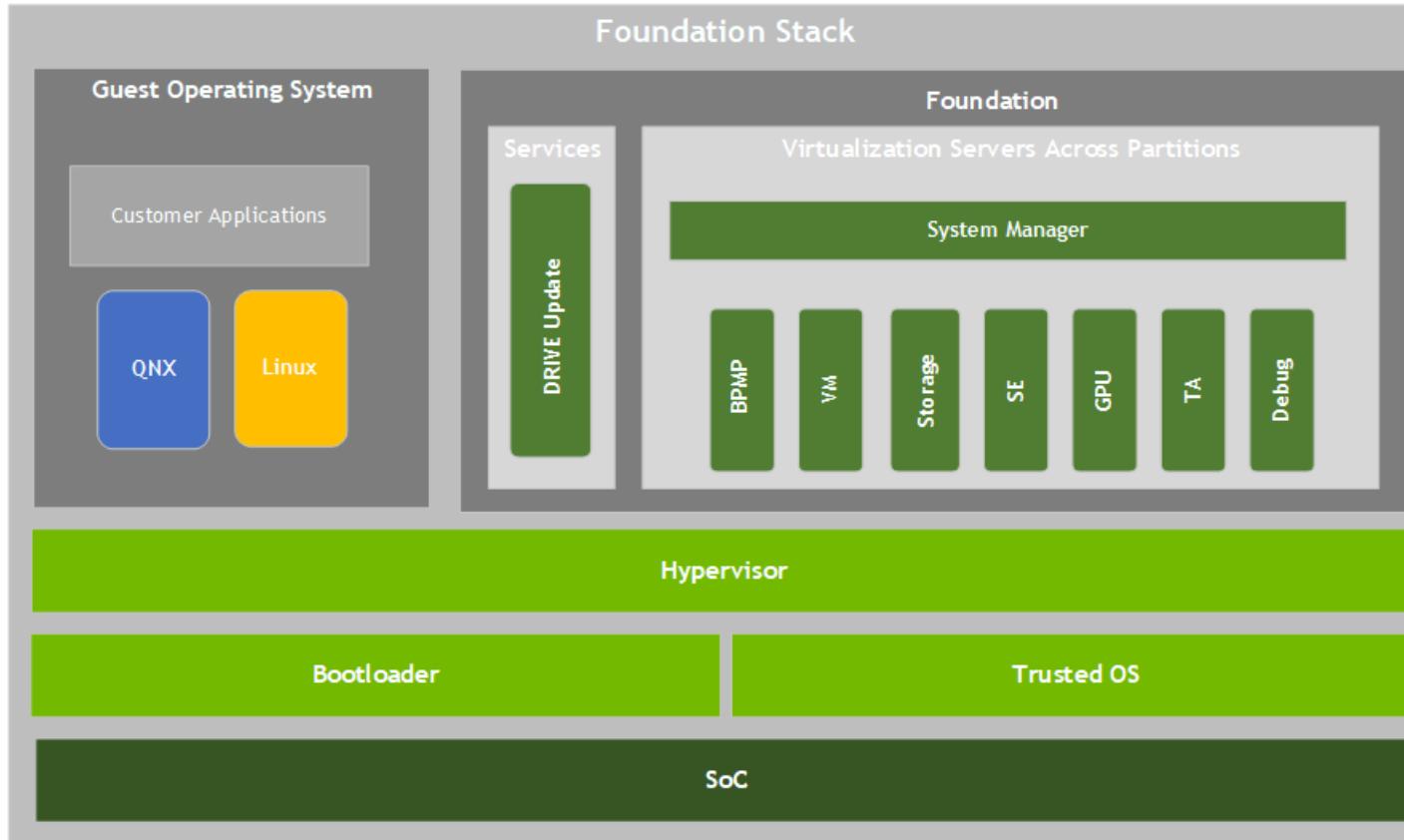
This section describes the platform component software stacks.

Use the DRIVE OS software stack to build your autonomous vehicle applications.



1.1.1 Foundation Services Stack

The NVIDIA DRIVE AGX™ platform Foundation services runtime software stack provides the infrastructure for all the components of the platform. With this infrastructure, guest OS systems can run on the hardware, with the Hypervisor managing use of hardware resources.



Foundation Components

Component	Description
Hypervisor	<p>Trusted Software server that separates the system into partitions. Each partition can contain an operating system or a bare-metal application. The Hypervisor manages:</p> <ul style="list-style-type: none"> > Guest OS partitions and the isolation between them. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Note: Multiple Guest OSs are not supported. In addition, you can run QNX or Linux, but not both. </div> <ul style="list-style-type: none"> > Partitions' virtual views of the CPU and memory resources. > Hardware interactions > Run-lists > Channel recovery <p>Hypervisor is optimized to run on the ARMv8.2 Architecture.</p>
guest OS	Allocates peripherals that Guest OS needs to control.
Services	Services for DRIVE Update.
<u>Bootloader</u>	Firmware that runs during boot to load firmware components, such as boot images, partition images, and other firmware.
Orin SoC	System on a Chip hardware resources.

Virtualized Servers

The virtualized configurable servers are as follows.

Component	Description
Boot and Power Manager Processor (BPMP) Server	Facilitates communication between Guest Virtual Machines (VM) and BPMP firmware.
VM Server	host1X virtualizes NvHost.

Component	Description
Security Engine (SE) Server	Para-virtualizes and allows multiple Guest Virtual Machines access to the security engine cryptographic hardware accelerator.
Trusted Applications (TA)	Through Trusted Applications, Trusted OS exposes a set of core services that use managed security assets in cryptographic operations without exposing them to non-secure guest software.
GPU	Runs on top of the virtualization core and handles sharing of GPU between multiple client Guest Virtual Machines. The GPU virtualization also includes the para-virtualized GPU client driver running inside of each Guest Virtual Machine.
Storage Server	Para-virtualizes storage access to enable sharing physical storage devices among multiple Guest Virtual Machines.
Debug Server	Provides support for kernel-level debugging of Guest Virtual Machines (VM).

1.1.2 NvMedia Architecture

NvMedia provides powerful processing of multimedia data for true hardware acceleration across NVIDIA DRIVE® Orin™ devices. With the NvMedia and Orin firmware components, multimedia applications support multiple simultaneous camera feeds for simultaneous processing. The NvMedia features include:

- Robust image processing and sensor control blocks for use by applications.
- Hardware surface to handle all types of image formats such as RGB, YUV and RAW.
- Functional components for image capture, processing, acceleration, encoding, and interoperating with other libraries.

Applications leverage the NvMedia Application Programming Interface (API) to process image and video data. Additionally, NvMedia can route image data to/from other components, such as OpenGL ES and NVIDIA® CUDA®.

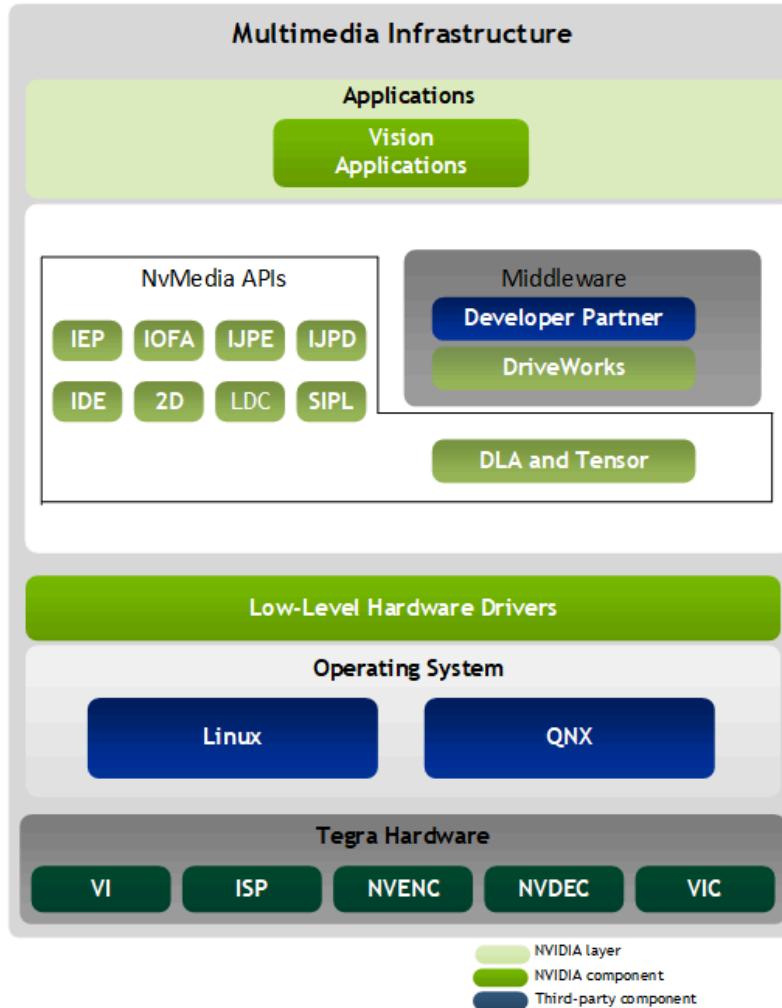
1.1.2.1 NvMedia Stack

The NvMedia software stack supports these types of interactions:

- Applications call the NvMedia Framework components to string together a sequence of processing steps for images.

- > The NvMedia Framework calls low-level hardware drivers to interact with the SoC components on the SoC chip.

Consult [building and running NvMedia sample applications](#) to build an NvMedia sample application.



Each component functionality is as follows. The drivers expose a subset of the functionality available.

Tegra Hardware	Description
Video Input (VI)	Receives CSI data from the camera.
Image Signal Processor (ISP)	Produces a processed image from image data captured from an image sensor. For example, it can make pixel-level changes such as inverting pixel bits, auto exposure, and white balance correction.

Tegra Hardware	Description
NVIDIA Encoder (NVENC)	Converts raw image data into one of the supported image formats.
NVIDIA Decoder (NVDEC)	Converts encoded image data into raw image data.
Video Interlace Compositor (VIC)	Converts video data for deinterlacing, composition, and format conversion.
Optical Flow Accelerator (OFA)	Accelerate optical flow and stereo disparity computation between the frames.

1.1.2.2 NvMedia Components

The NvMedia components are as follows:

Image Component	Description
SIPL	The SIPL framework processes incoming image data with AE, AWB controls and includes the ISP processing capability. It supports camera tuning tools.
2D	Image 2D provides the ability to perform manipulation of image data; such as cropping, scaling, copying, and converting the format.
IEP	Image Encode Processing provides the ability to encode processed YCRCb surface inputs to H.264 and H.265.
IJE	Image JPEG Encoding provides the ability to encode YUV surfaces to JPEG format.
IJD	Image JPEG Decoding provides the ability to decode images compressed in JPEG format to raw YUV surfaces.
LDC	Image LDC provides the ability to perform lens distortion correction and temporal noise reduction on image data.

1.1.2.3 NvMedia APIs and Thread Safety

NvMedia APIs are not designed to be thread safe. It is the responsibility of the application to maintain thread safety. To ensure thread safety:

- NvMedia components can be created and used in any thread but the APIs cannot be used from different threads concurrently.
- Different instances of the same component can be used in parallel from different threads.
- Encoders are designed to be fed from one thread and get the encoded bitstream from another thread.

Chapter 2. Installation

The following chapters describe how to:

- Set up Docker and NVIDIA GPU Cloud access
- Set up DRIVE OS with Debian packages
- Build and run samples

2.1 Install DRIVE OS Linux Docker Containers from NGC

2.1.1 Set Up Docker and NVIDIA GPU Cloud Access



Note: The minimum required version of Docker Engine to run DRIVE OS Linux Docker containers is 19.03.

- On the host system, download the type of Docker software needed for your organization (Enterprise, Desktop, or other) from <https://www.docker.com/>.
- Alternatively, download an installation script from <https://get.docker.com/> and install DRIVE OS Docker on your host system with a command similar to the following:

```
wget -qO- https://get.docker.com/ | ${SHELL}
```

2.1.1.1 NVIDIA GPU Cloud Access

In order to access [NVIDIA GPU Cloud \(NGC\)](#), you will need an [NVIDIA Developer Account](#), and membership in the [NVIDIA DRIVE® Developer Program for DRIVE AGX](#). Please register for an account and apply for membership in the Developer Program before proceeding.



Note: It may take up to a few days for access to the Developer Program to be approved.



Note: Please reference [NVIDIA DRIVE Platform Docker Containers](#) for more information on DRIVE OS Docker Container workflows.

2.1.1.2 Sign in to NVIDIA GPU Cloud



Note: Please make sure you've received the NGC activation email and successfully activated your NGC Organization(s) and NGC Team(s) before proceeding.

1. On the host system, sign into NGC (<https://ngc.nvidia.com>) using your NVIDIA Developer credentials.
2. Once you have signed in, select your NGC Organization (for NVIDIA Developer Users, this will be drive). DRIVE OS Docker Containers are located under PRIVATE REGISTRY.

The screenshot shows the NVIDIA NGC | PRIVATE REGISTRY interface. On the left, a sidebar menu is open, highlighting the "PRIVATE REGISTRY" section with a green oval. The menu includes options like Entity Creation Hub, Collections, Containers, Helm Charts, Models, and Resources. Below this is the "ORGANIZATION" section. At the bottom of the sidebar are icons for Help, Edit, and Collapse, along with the text "NVIDIA DRIVE OS Linux SDK Developer Guide" and "NGC Version: v2.99.1".

Private Registry > Containers

Containers

ALL CONTAINERS

[Xavier] DRIVE OS Linux SDK Flash Container

NVIDIA DRIVE® OS Linux SDK Docker container for flashing advanced Autonomous Vehicles software that runs on NVIDIA...

[View Labels](#) [Copy Image Path](#)

ORGANIZATION

[Xavier] DRIVE OS Linux SDK Flash Container

NVIDIA DRIVE® OS Linux SDK Docker im... Autonomo... runs on N...

[View Labels](#)

3. Once signed in, select **Setup** under the User menu in the top-right of the page to Generate an API Key to pull Docker images.

NVIDIA NGC | SETUP

- CATALOG
- PRIVATE REGISTRY
- ORGANIZATION

?

-collapse

NGC Version: 2.69.2

Setup

Setup

[Generate API Key](#)



Generate your own API key to use the NGC service through the Docker client.

[Get API Key](#)

For more information, see:

<https://docs.nvidia.com/ngc/ngc-overview/index.html#generating-api-key>



Note: Using NVIDIA GPU Cloud is beyond the scope of this document. Please refer to the [NVIDIA GPU Cloud documentation](#) for more information.

4. Log in to the NGC container registry.

```
sudo docker login nvcr.io
```

5. When prompted for your username, enter the following text:

```
$oauthtoken
```

The \$oauthtoken username is a special username that indicates that you will authenticate with an API key, not a username and password.

6. When prompted for your password, enter your NGC API key as shown in the following example:

```
Username: $oauthtoken
```

```
Password: my-api-key
```

See [Docker Login documentation](#) for more information on login methods.

After you log in, you have access to NVIDIA DRIVE OS Docker images, depending on your specific permissions in the registry.

2.1.2 Set Up DRIVE OS Linux with NVIDIA GPU Cloud (NGC)

Begin by referring to the instructions in [Setup Docker and NVIDIA GPU Cloud Access](#), then proceed with the following installation guide for DRIVE OS Linux.

2.1.2.1 Images Available in This Release

The following images will be available on NGC in this release:

File Name	Intent
drive-agx-orin-linux-aarch64-sdk-build-x86:latest	Build and Flash DRIVE OS 6.0.8 SDK Linux

2.1.2.2 Pull Docker Images from NGC



Note: Prior to installation, you can choose to remove previously installed DRIVE OS Docker images/containers to increase space capacity.

2.1.2.2.1 Instructions for Running DRIVE OS Docker

1. Log into the NVIDIA GPU Cloud (NGC) using instructions in [the previous section](#).
2. On the host system, pull the image using the following command:



Note: \${MY_NGC_ORG} is the NGC Organization allocated for your team.

```
sudo docker run -it --privileged --net=host -v /dev/bus/usb:/dev/bus/usb -v  
${WORKSPACE}:/home/nvidia/  
nvcr.io/${MY_NGC_ORG}/driveos-sdk/drive-agx-orin-linux-aarch64-sdk-build-x86:latest
```

2.1.2.3 Flash DRIVE OS Linux

Use the procedures in this section to flash NVIDIA DRIVE™ OS Linux to the target system from the Docker container.

2.1.2.3.1 Flash Using the DRIVE OS Docker Container

1. Log on to the NVIDIA GPU Cloud (NGC). If you need to download a DRIVE OS Docker image, use the procedures in the [Set Up Docker and NVIDIA GPU Cloud Access](#) section of this document.
2. Connect NVIDIA DRIVE™ AGX to the host system.



Note: Ensure that NVIDIA DRIVE AGX is connected to the host system, and that no other processes, such as TCUMuxer or Minicom are holding a lock on /dev/ttyACM* before starting the Docker container.

3. Start the DRIVE OS Docker container and flash DRIVE OS onto the NVIDIA DRIVE AGX using the following command.

```
sudo docker run -it --privileged --net=host -v /dev/bus/usb:/dev/bus/usb -v /  
drive_flashing:/drive_flashing  
nvcr.io/${MY_NGC_ORG}/driveos-sdk/drive-agx-orin-linux-aarch64-sdk-build-x86:latest
```

2.1.2.3.1.1 Instructions for Flashing

Flash with flash.py:

```
./flash.py /dev/ttyACM1 p3710
```



Note: /dev/ttyACM1 is the default port for the DRIVE AGX Orin Developer Kit on the x86 Host system if only one Developer Kit or another USB device is connected. If there are additional devices in this device range, please disconnect them before flashing.



Note: If flashing the target system fails, try the following:

1. Reseat the USB cable connected from the Host PC to the LEFT USB Type-C port on the target on both ends. See the "NVIDIA DRIVE AGX Orin Developer Kit Hardware Quick Start Guide" for more details.
2. Use a USB 3.1 port on the host PC if available.
3. Use a different USB C/SS cable, OR a higher speed cable such as a USB SS10 (for example, Amazon Model Number L6LUC146-CS-R).
4. Use a USB 2.0 port on the host PC used for flashing.

2.1.2.3.1.2 Browse DRIVE OS Documentation

The DRIVE OS Developer Guide can be browsed from within Docker. To do so, start the Docker container using the instructions above for running DRIVE OS Docker.

Once in the container, execute the following to start the documentation server.

```
# start-docs-server
```

This will start a lightweight HTTP server on port 65535 by default. If the port is not available, you can configure the process to start on a different port. The below example will attempt to use port 49350 for the HTTP server, if it is available.

```
# start-docs-server 49350
```

Once the server starts, use your preferred browser to access the docs by visiting <http://localhost:65535> (replacing 65535 with your specified port, if different). If you have started the container on a remote machine, you will need to use the remote IP as opposed to localhost.

The process can be stopped with CTRL+C.

2.2 Set Up DRIVE OS with Debian Packages

2.2.1 Prerequisites

Prior to the installation of DRIVE OS local repo Debian packages, ensure the Ubuntu host apt database is clear of any errors. The below command will return a non-ZERO value in case of issues. Please make sure to correct these issues before proceeding with the installation steps in this guide.

```
sudo apt update
echo $?
```



Caution: Proceeding with a non-ZERO value can result in installation errors.

2.2.2 Downloading from NVONLINE

1. Log into NVONLINE (partners.nvidia.com) with your NVIDIA Partner Account. If you do not have an NVONLINE account, please contact your NVIDIA representative.
2. To access the Debian packages, find the **DRIVE OS 6.0.8 Linux SDK** group for Linux.
3. The downloaded local repo will have the local repo Debian package and extra Debian packages that are available to be installed outside of the local repo.
4. The local repo Debian package has the following format:

```
nv-driveos-repo-[SDK]-[OS]-[RELEASE]-[BUILD]-[GCID]_amd64.deb
```

Select the **Download All** button to download all Debian packages simultaneously.

DRIVE OS Linux Debian Package File List

Package	Filename	Description
nv-driveos-repo-sdk-linux-*	nv-driveos-repo-sdk-linux-*_amd64.deb	Installs NVIDIA DRIVE OS Linux SDK/PDK/Flash Components Installs all DRIVE OS Linux Components provided by NVIDIA
cuda-repo-ubuntu*-local	cuda-repo-ubuntu*-local_*_amd64.deb	CUDA repository configuration files Contains repository configuration for CUDA. Contains a local repository for CUDA.
nv-driveos-linux-vksc-dev-*	nv-driveos-linux-vksc-dev-*_amd64.deb	Installs NVIDIA DRIVE OS Linux VKSC DEV module Installs all DRIVE OS Linux VKSC DEV components provided by NVIDIA

Package	Filename	Description
nv-driveos-linux-config-dhcp-*	nv-driveos-linux-config-dhcp-*_amd64.deb	Installs NVIDIA DRIVE OS LINUX DHCP config Installs all DRIVE OS LINUX DHCP components provided by NVIDIA
driveworks-stm	driveworks-stm_*~linux*_arm64.	NVIDIA DriveWorks software development kit (SDK) provides a foundation to build applications for autonomous driving, providing computationally intensive algorithms for object detection, map localization, and path planning.
driveworks-cross	driveworks-cross_*~linux*_amd6	NVIDIA DriveWorks software development kit (SDK) provides a foundation to build applications for autonomous driving, providing computationally intensive algorithms for object detection, map localization, and path planning.
nv-tensorrt-repo-ubuntu2004-cuda*-tr*-d61-target-ga-*	nv-tensorrt-repo-ubuntu2004-cuda*-tr*-d61-target-ga-*_arm64.deb	nv-tensorrt repository configuration files Contains repository configuration for nv-tensorrt. Contains a local repository for nv-tensorrt.
nv-driveos-linux-driveos-oobe-desktop-ubuntu-20.04-rfs-*	nv-driveos-linux-driveos-oobe-desktop-ubuntu-20.04-rfs-*_amd64.deb	NVIDIA DRIVE OS Linux Filesystem image file
nv-driveos-linux-driveos-oobe-ubuntu-20.04-rfs-*	nv-driveos-linux-driveos-oobe-ubuntu-20.04-rfs-*_amd64.deb	NVIDIA DRIVE OS Linux Filesystem image file
nv-driveos-linux-config-minicom-*	nv-driveos-linux-config-minicom-*_amd64.deb	Installs NVIDIA DRIVE OS Linux minicom config Installs all DRIVE OS Linux minicom components provided by NVIDIA
driveworks-stm-cross	driveworks-stm-cross_*~linux*_amd6	NVIDIA DriveWorks software development kit (SDK) provides a foundation to build applications for autonomous driving, providing computationally intensive algorithms for object detection, map localization, and path planning.
nv-driveos-linux-yocto-*	nv-driveos-linux-yocto-*_amd64.deb	Installs NVIDIA DRIVE OS Linux YOCTO Components Installs all DRIVE OS Linux YOCTO Source Components provided by NVIDIA

Package	Filename	Description
nv-driveos-linux-yocto-oss-src-*	nv-driveos-linux-yocto-oss-src-*_amd64.deb	<p>Installs NVIDIA DRIVE OS Linux Yocto OSS Source SDK Components</p> <p>Installs all DRIVE OS Linux Yocto OSS Source Components provided by NVIDIA</p>
nsight-graphics-for-embeddedlinux-pro-*	NVIDIA_Nsight_Graph	NVIDIA Nsight Graphics is a standalone application for the debugging, profiling, and analysis of graphics applications.
driveworks-samples	driveworks-samples_*~linux*_ar	NVIDIA DriveWorks software development kit (SDK) provides a foundation to build applications for autonomous driving, providing computationally intensive algorithms for object detection, map localization, and path planning.
driveworks-cgf-cross	driveworks-cgf-cross_*~linux*_amd6	NVIDIA DriveWorks software development kit (SDK) provides a foundation to build applications for autonomous driving, providing computationally intensive algorithms for object detection, map localization, and path planning.
cuda-repo-cross-aarch64-ubuntu*-local	cuda-repo-cross-aarch64-ubuntu2004-*_all.deb	<p>CUDA repository configuration files</p> <p>Contains repository configuration for CUDA.</p> <p>Contains a local repository for CUDA.</p>
cuda-tegra-repo-ubuntu*-local	cuda-tegra-repo-ubuntu2004-*_arm64.deb	<p>cuda-tegra repository configuration files</p> <p>Contains repository configuration for cuda-tegra.</p> <p>Contains a local repository for cuda-tegra.</p>
nv-driveos-linux-ubuntu-20.04-src-*	nv-driveos-linux-ubuntu-20.04-src-*_amd64.deb	<p>Ubuntu Sources for NVIDIA DRIVE OS Linux filesystem.</p> <p>Provides Ubuntu Sources that are used to build NVIDIA DRIVE OS Linux filesystem.</p>
nv-tensorrt-repo-ubuntu2004-cuda*-trt*-x86-host-ga-*	nv-tensorrt-repo-ubuntu2004-cuda*-trt*-x86-host-ga-*_amd64.deb	<p>nv-tensorrt repository configuration files</p> <p>Contains repository configuration for nv-tensorrt.</p> <p>Contains a local repository for nv-tensorrt.</p>
nv-driveos-linux-config-nfs-*	nv-driveos-linux-config-nfs-*_amd64.deb	<p>Installs NVIDIA DRIVE OS Linux NFS config</p> <p>Installs all DRIVE OS Linux NFS components provided by NVIDIA</p>

Package	Filename	Description
nv-driveos-linux-vksc-ecosystem-*	nv-driveos-linux-vksc-ecosystem-*_amd64.deb	<p>Installs NVIDIA DRIVE OS Linux VKSC ECOSYSTEM module</p> <p>Installs all DRIVE OS Linux VKSC ECOSYSTEM components provided by NVIDIA</p>
nv-driveos-linux-tegra2aurix-updater-*	nv-driveos-linux-tegra2aurix-updater-*_amd64.deb	<p>Installs NVIDIA DRIVE OS Linux Tegra2Aurix updater hex files</p> <p>Installs all DRIVE OS Linux Tegra2Aurix updater hex files provided by NVIDIA</p>
nv-tensorrt-repo-ubuntu2004-cuda*-trt*-d61-cross-ga-*_amd64.deb	nv-tensorrt-repo-ubuntu2004-cuda*-trt*-d61-cross-ga-*_amd64.deb	<p>nv-tensorrt repository configuration files</p> <p>Contains repository configuration for nv-tensorrt</p> <p>Contains a local repository for nv-tensorrt</p>
nv-driveos-linux-ubuntu-20.04-base-*	nv-driveos-linux-ubuntu-20.04-base-*_amd64.deb	<p>Installs Canonical's Base Ubuntu filesystem.</p> <p>Provides Canonical's Base Ubuntu filesystem.</p>
driveworks-stm-samples	driveworks-stm-samples_*~linux*_arm64.deb	NVIDIA DriveWorks software development kit (SDK) provides a foundation to build applications for autonomous driving, providing computationally intensive algorithms for object detection, map localization, and path planning.
driveworks	driveworks_*~linux*	NVIDIA DriveWorks software development kit (SDK) provides a foundation to build applications for autonomous driving, providing computationally intensive algorithms for object detection, map localization, and path planning.
nv-driveos-linux-mlnx-docker-arm64-debians-*	nv-driveos-linux-mlnx-docker-arm64-debians-*_amd64.deb	<p>Installs NVIDIA Docker and Mellanox filesystem Debians</p> <p>Installs the NVIDIA Docker and Mellanox filesystem Debians required to rebuild the filesystems</p>
driveworks-cgf-samples	driveworks-cgf-samples_*~linux*_arm64.deb	NVIDIA DriveWorks software development kit (SDK) provides a foundation to build applications for autonomous driving, providing computationally intensive algorithms for object detection, map localization, and path planning.
driveworks-cgf	driveworks-cgf_*~linux*_arm64.deb	NVIDIA DriveWorks software development kit (SDK) provides a foundation to build applications for autonomous driving, providing computationally intensive algorithms for object detection, map localization, and path planning.

Package	Filename	Description
nv-driveos-linux-p3898-specific-*	nv-driveos-linux-p3898-specific-*_amd64.deb	<p>Installs NVIDIA DRIVE OS Linux SDK components specific for the P3898 platform</p> <p>Installs all DRIVE OS Linux SDK P3898 platform specific components provided by NVIDIA</p>
nv-driveos-linux-ubuntu-20.04-arm64-debians-*	nv-driveos-linux-ubuntu-20.04-arm64-debians-*_amd64.deb	<p>Installs NVIDIA DRIVE OS Linux filesystem ARM64 Debians</p> <p>Installs all DRIVE OS Linux filesystem ARM64 Debians provided by NVIDIA required to rebuild the filesystems</p>
nv-driveos-repo-pdk-linux-*	nv-driveos-repo-pdk-linux-*_amd64.deb	<p>Installs NVIDIA DRIVE OS Linux SDK/PDK/Flash Components</p> <p>Installs all DRIVE OS Linux Components provided by NVIDIA</p>
nv-driveos-linux-target-arm64-debians-*	nv-driveos-linux-target-arm64-debians-*_amd64.deb	<p>Installs DRIVE OS Linux target Debians for standard filesystems</p> <p>Installs the DRIVE OS Linux target Debians for standard filesystems required for rebuilding standard filesystems.</p>

2.3 Install DRIVE OS Linux Debian Packages

2.3.1 Uninstall Steps for Local Repo Debian Packages

If you have previously installed DRIVE OS, the existing DRIVE OS first needs to be uninstalled. This will help maintain content integrity and avoid any conflicts.

To remove or uninstall, please follow these instructions:

```
sudo -E apt-get -y --purge remove "nv-driveos*"
```

```
sudo apt-get -y autoremove
```

Optional step: The DRIVE OS Debian Package uninstall will not remove any data from \$NV_WORKSPACE.

If you want to remove, please follow the below instruction:

```
sudo rm -rf $NV_WORKSPACE
```

2.3.2 Installation Steps for Local Repo Debian Packages

Instructions for installing local repo Debian packages.

1. To install the local repo Debian packages, use dpkg.

```
sudo dpkg -i ./nv-driveos-repo-sdk-linux-[VERSION]-[GCID]_[VERSION]_amd64.deb
```

2. Install top-level DRIVE OS SDK Debian packages:

```
export NV_WORKSPACE=/path/where/SDK/needs/to/beinstalled
```

If using SDK Local Repo Debian packages:

```
sudo -E apt -f -y install nv-driveos-build-[SDK]-[OS]-[RELEASE]-[GCID]
2>&1 | tee nv-driveos-build-[SDK]-[OS]-[RELEASE]-[GCID]_install.log
```

After a successful installation of nv-driveos-build-[SDK]-[OS]-[RELEASE]-[GCID], the SDK install directory structure will be as following:

\$NV_WORKSPACE directory structure	Sub-directories
drive-foundation	firmware make platform-config security tools version-nv-sdk.txt virtualization
drive-linux	filesystem firmware include kernel lib-target make oss samples tools tests
drive-linux_src	3rdparty_dtc_src.tgz e2fsprogs-1.41.11.tar.gz jq-1.5.tar.gz kernel linuxptp.tgz NVIDIA- kernel-module-source-TempVersion.tar.xz wayland yocto
toolchains	aarch64--glibc--stable-2022.03-1 armv5-eabi-- glibc--stable-2020.08-1 armv7-eabihf--glibc-- stable-2020.08-1

After exporting NV_WORKSPACE, the following commands should be run:

```
sudo apt update
sudo apt upgrade
```

2.3.2.1 Installation Steps for the Extra Packages

1. Install the below CUDA 11.4 Debian packages for Linux available under the same folder.

```
sudo dpkg -i ./cuda-repo-ubuntu2004-11-4-local_[CUDA-VERSION]-[DRIVER-VERSION]-1_amd64.deb
sudo dpkg -i ./cuda-repo-cross-aarch64-ubuntu2004-11-4-local_[CUDA-VERSION]-1_all.deb
sudo apt-key add /var/cuda-repo-ubuntu2004-11-4-local/[filename].pub
sudo apt-key add /var/cuda-repo-cross-aarch64-ubuntu2004-11-4-local/[filename].pub
sudo apt update
sudo apt -y install cuda-toolkit-11-4 -y
sudo apt -y install cuda-cross-aarch64-11-4 -y
```

2. In case of any issue with the installation, remove old packages and reinstall.

```
sudo rm /var/lib/apt/lists/_var_cuda*
sudo apt --fix-broken install -y
sudo apt autoremove -y
sudo apt remove--purge -y "cuda*"
sudo apt remove--purge -y "*cublas*"
sudo apt remove--purge -y "*nsight*"
sudo apt autoremove -y
```

3. Install the cuDNN Debian packages for Linux:

```
sudo apt install ./cudnn-local-repo-ubuntu2004-[CUDNN-VERSION].deb
sudo apt-key add /var/cudnn-local-repo-ubuntu2004-[CUDNN-VERSION]/[filename].pub
sudo apt update
sudo apt install libcudnn8 -y
sudo apt install libcudnn8-dev -y
sudo apt install libcudnn8-samples -y
```

4. Install the cuDNN Debian packages for cross-compiling on Linux:

```
sudo apt install ./cudnn-local-repo-cross-aarch64-ubuntu2004-[CUDNN-VERSION].deb
sudo apt-key add /var/cudnn-local-repo-cross-aarch64-ubuntu2004-[CUDNN-VERSION]/[filename].pub
sudo apt update
sudo apt install libcudnn8-cross-aarch64 -y
```

5. Install TensorRT Debian packages for Linux:

```
sudo dpkg -i nv-tensorrt-repo-ubuntu2004-cuda11.4-trt[RELEASE]-x86-host-[ea|ga]-[BUILD-DATE]_1-1_amd64.deb
sudo apt-key add /var/nv-tensorrt-repo-ubuntu2004-cuda11.4-trt[RELEASE]-x86-host-[ea|ga]-[BUILD-DATE]/[filename].pub
sudo apt update
sudo apt install tensorrt -y
```

6. Install TensorRT Debian packages for cross-compiling on Linux:

```
sudo dpkg -i nv-tensorrt-repo-ubuntu2004-cuda11.4-trt[RELEASE]-d61-cross-ga-20221229_1-1_amd64.deb
sudo apt-get install tensorrt-cross-aarch64 -y
```

P3898-Specific Package Installation

If you are using a P3898 platform (see [DRIVE Platform Supported Boards](#) for more information), install the below platform-specific package for DRIVE OS Linux:

```
sudo -E apt install ./nv-driveos-linux-p3898-specific
```

If there are any issues with the installation, remove old packages and reinstall with the following commands:

```
sudo -E apt-get -y --purge remove "nv-driveos-linux-p3898-specific"
sudo apt-get -y autoremove
```

2.3.3 Flash DRIVE OS Linux

2.3.3.1 Instructions for BIND PCT

```
cd $NV_WORKSPACE/drive-foundation
./make/bind_partitions -b <board-variant> linux
```



Note: Please refer to [DRIVE Platform Supported Boards](#) to identify your board variant.

2.3.3.2 Instructions for Flashing

1. Put the DRIVE AGX system into reset mode from the Aurix MCU console.

Connect to the Aurix MCU console:

```
sudo minicom -w -D /dev/ttyACM1
```

From the Aurix MCU console:

```
tegrarecovery x1 on
tegrareset x1
```

2. Flash:

```
tools/flashtools/bootburn/bootburn.py -b <board-variant>
```



Note: You do not need to use sudo for the following commands:

- > bootburn.py
- > create_bsp_images.py
- > flash_bsp_images.py

However, if you choose to use sudo for one, use it consistently with the others. Do not switch between sudo/non-sudo usage.



Note: Please refer to [DRIVE Platform Supported Boards](#) to identify your board variant.

2.4 Download and Run SDK Manager

Instructions for downloading and running SDK Manager from [NVIDIA Developer](#) are detailed below.

2.4.1 Download via NVIDIA DRIVE Developer Program

The following instructions are for NVIDIA DRIVE Developer Program users.

1. Navigate to <https://developer.nvidia.com/nvidia-sdk-manager>, and log in.

Alternatively, you can access SDK Manager from the download page that corresponds with the product category you are installing (for example, [NVIDIA DRIVE Downloads](#)).

2. From the download page, locate the Debian package for Ubuntu or download directly from https://developer.nvidia.com/sdkmanager_deb.
3. Download the file to your host machine.

2.4.2 Install the SDK Manager Package



Note: Ubuntu version 20.04 LTS comes with Python3 installed as the default and /usr/bin/python is not present, as detailed here: https://wiki.ubuntu.com/FocalFossa/ReleaseNotes#Other_base_system_changes_since_18.04_LTS. However, DRIVE OS installation using SDK Manager requires /usr/bin/python in the system. To address this, run the following command before starting the installation:

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3 1
```

Alternatively, you can install the python-is-python3 package, which will cause /usr/bin/python to point to python3 instead:

```
sudo apt install python-is-python3
```

Once you have downloaded the SDK Manager .deb file to your host machine, do the following.

1. From a terminal, install the Debian package:

```
sudo apt install ./sdkmanager_[version]-[build#]_amd64.deb
```

2. Next, you can start SDK Manager using one of the following two methods:
 - a. Launch SDK Manager from the Ubuntu launcher.

- b. Open a terminal and launch SDK Manager with the following command:

```
sdkmanager
```



Note: SDK Manager also supports a command line interface. To see the options, run:

```
sdkmanager --help
```

To learn more, see [Command Line Install](#).

2.4.3 Log In to SDK Manager

1. From the SDK Manager launch screen, select the appropriate login tab for your account type and installation.
 - > NVIDIA DRIVE Developer Program — developer.nvidia.com
 - > Offline — to install SDKs that were previously downloaded, and are available from a local folder or mounted drive. For more information, see [Offline Install](#).

The default login tab is for **NVIDIA Developer**.

- a. On the SDK Manager log in page, enter the credentials for your NVIDIA Developer account, and click **Login**.
- b. Once completed, SDK Manager will start.



The image shows the NVIDIA SDK Manager login interface. The background features a dark, geometric, crystalline pattern. At the top left, the text "SDK Manager" is visible. In the center, the words "NVIDIA SDK M" are partially visible. On the right side, there are two buttons: "NVIDIA DEVELOPER developer.nvidia.com" and "NVONLINE partners.nvidia.com". Below these buttons, the text "NVIDIA DEVELOPER LOGIN" is displayed in large green letters. A sub-instruction "Click LOGIN to initiate login process in your default browser." is present, followed by the note "Manager will start once done." A large green "LOGIN" button is centered at the bottom. Below it is a checkbox labeled "Stay logged in" with a checked status.

SDK Manager

NVIDIA DEVELOPER
developer.nvidia.com

NVONLINE
partners.nvidia.com

NVIDIA DEVELOPER LOGIN

Click LOGIN to initiate login process in your default browser.
Manager will start once done.

Stay logged in



NVIDIA. Copyright © 2020, NVIDIA CORPORATION. All rights reserved. | NVIDIA Developer

QR code login option:

- a. Click the QR code icon from the sign in panel.
- b. When the QR code image appears, scan the code via a camera application on a different device.
- c. Enter the credentials for your NVIDIA Developer account on that device.
- d. Once completed, SDK Manager will start.



2. Before proceeding, choose whether or not to enable data collection.

2.5 Finalize DRIVE AGX Orin System Setup



Note: If the End User License Agreement (EULA) has not been accepted on the DRIVE AGX Orin Developer Kit, the DP display will not be activated when flashing is completed. The user *must* connect to the DRIVE AGX platform console via a terminal emulator to determine when flashing has been completed, and to complete the platform setup.

Instructions for connecting a terminal emulator to the platform are in the NVIDIA DRIVE OS Linux SDK Developer Guide sections [Using tcu_muxer](#) and [Terminal Emulation](#). You can connect the Developer Kit using the following command:

```
sudo minicom -w -D /dev/ttyACM(0/1)
```



Note: The username, password, and security profile/setup will remain persistent on later flashes once they are set the first time after flashing the DRIVE OS SDK release. The user will not be prompted for this information again when flashing this or later releases. However, the user may need to clear the persistent data when downgrading to an earlier release. See the NVIDIA DRIVE OS Linux SDK Developer Guide section on [Methods to Reset Persistence Partition](#) for information on removing the persistent data, and User Management for information on adding/deleting/changing users or changing the security profile and settings.

1. Accept EULA and Set Admin Username/Password.

See the NVIDIA DRIVE OS Linux SDK Developer Guide section on [DRIVE OS Linux oem-config](#) for more information on these prompts. (This is only needed for the first boot out of the box, and does not apply to SDK Manager or Docker flashed systems, unless persistent data is cleared.)

2. Select SSH Profile and Other Setup Options.

DRIVE OS Linux provides two profiles for security setup (including SSH): an NVIDIA enhanced security profile that uses ECDSA-based algorithms for SSH security, or stock Ubuntu 20.04 configuration. The customer must select the profile on the first setup screen. Both profiles will enable SSH server in the target by default.

The user can follow the other prompts after SSH profile selection to install additional users; see [DRIVE OS Linux oem-config](#) for more information [not prompted if already

set on DRIVE OS Linux first flashing]. After these prompts are completed, the platform will boot to the prompt to enter the username.



Note: If you flash with NGC Dockers with the bootburn --init_persistent_partitions option, the new login and password is not set on the target on flashing, and the expected system configuration screens do not appear. You can still add/delete user logins or change passwords using the information in the NVIDIA DRIVE OS Linux SDK Developer Guide section on [DRIVE OS User Management](#).



Note: You do not need to use sudo for the following commands:

- > bootburn.py
- > create_bsp_images.py
- > flash_bsp_images.py

However, if you choose to use sudo for one, use it consistently with the others. Do not switch between sudo/non-sudo usage.

2.6 Getting Started with DRIVE OS 6.x Linux Development

This guide is for the users of the DRIVE OS Linux operating system running on the NVIDIA DRIVE AGX platforms.

2.6.1 Target SSH Access from the Linux Host

If using a network connection ("Preferred method"), ensure the target and host system are both connected to the same network. Both the target and the host will be assigned an IP address automatically.

If a direct connection between the host and the target is required, refer to the DRIVE OS 6.0 SDK Developer Guide [Setting Up Networking on the Host and Target](#) for more information about enabling the DHCP server.

As discussed in [Finalize DRIVE AGX Orin System Setup](#), during the initial setup of the target device, you are prompted to choose an SSH profile. If you choose the DRIVE OS Linux Secure Login profile, see the DRIVE OS 6.0 SDK Developer Guide section in [SSH Key-Based Authentication from Clients to Server](#) to create a key on the Linux host to connect to the target.

The DRIVE AGX Developer Kit is pre-flashed with the SSH server installed and enabled by default. If you flash the developer kit with SDK Manager or Docker, the SSH server is also installed and enabled by default. However, if you flash with the base DRIVE OS SDK Debian build, which uses the `driveos-core-rfs` filesystem, the SSH server will

not be installed by default. Therefore, you need to install and set up by following the instructions as described in [DRIVE OS Linux SSH Server](#).



Note: While ping and ifconfig are not installed by default in the DRIVE OS Docker, ssh client and rcp are, so the user will be able to ssh/rcp between the Docker and target using the following procedures.

To confirm the overall network and SSH installation, follow these steps:

1. Verify that the target can communicate to the host by using the ping command.

```
ping <target-IP>
```

2. Launch the ssh client.

```
ssh <user>@<target-IP>
```

Use the username and password that were set for the system setup during the initial boot process. Verify that the same terms are used here. Follow the steps in the software guides found at [developer.nvidia.com/drive/start](#).

If the ping or ssh command is not successful and the target does not respond, you can launch the UART terminal (Minicom) to access the target.

1. Ensure that only one DRIVE AGX platform is connected to the USB port on the host system.
2. On the host system, confirm the ttyACM0 and ttyACM1 are listed under the /dev/ folder.
3. Run Minicom.

```
minicom -D /dev/ttyACM0
```

4. Check the IP address.

```
ifconfig eqos_0
```



Note: eqos_0 is the interface name for the RJ45 interface on the lower center of the Developer Kit backpanel. If other Developer Kit Ethernet interfaces are used, the interface name will vary.

2.7 Build and Run Sample Applications for DRIVE OS 6.x Linux



Note: To access the GPU from a Docker Container, please ensure that you have NVIDIA Container Toolkit installed. Installation instructions for NVIDIA Container Toolkit can be found at <https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html> for your host development environment distribution. An NVIDIA GPU and appropriate CUDA drivers must also be available on the host to run GPU-accelerated applications, including samples.



Note: If you are compiling and running samples in Docker and you want to preserve the compiled samples, please keep in mind that Docker containers are a temporary environment that will essentially be deleted when the container is stopped. To preserve the changes made in a running container, please refer to the Docker official documentation on committing Docker images: <https://docs.docker.com/engine/reference/commandline/commit>.

Alternatively, if the container was started with a mounted \${WORKSPACE}, you may copy or move the compiled binaries to the mounted \${WORKSPACE} before exiting the container so that the compiled binaries will be available on the host.

2.7.1 Graphics Applications

For boards that are pre-flashed or flashed with the custom GDM desktop file system (default for SDK Manager and Docker installations), the Bubble sample application is available in the /opt/nvidia/drive-linux/samples/ folder.

To run a basic X11 sample, use the following commands:

```
$ cd /opt/nvidia/drive-linux/samples.opengles2/bubble/x11  
$ ./bubble -fps
```

To run other graphics samples for X11 and the supported window systems, see [Building and Running Samples](#) and [Window Systems](#).

2.7.2 CUDA

Documentation is available online: [CUDA Toolkit Documentation v11.4.1](#).

- > The installations of CUDA Host x86 and Linux AArch64 are in the /usr/local/cuda-11.4/ folder.
- > The source code files of all CUDA samples are in the /usr/local/cuda-11.4/samples folder.

2.7.2.1 How to Build the CUDA Samples for the Linux Target

Host cross-compile is supported for DRIVE OS releases only. After you finish installing CUDA x86 and cross-compile packages, perform the following steps:

1. Install the CUDA sample sources to a directory on your x86 development host where you do not need root privileges to write, such as the \$HOME directory as shown in the following example:

```
$ cd ~/
```

- > Use the following command to run the cuda-install-samples-11.4.sh script from the CUDA installation in the x86 host file system:

```
$ <$NV_WORKSPACE>/drive-linux/filesystem/targetfs/usr/local/cuda-11.4/bin/cuda-
install-samples-11.4.sh .
```

Where \$NV_WORKSPACE is:

- Docker: /drive

- > A successful run will show the following:

```
Copying samples to ./NVIDIA_CUDA-11.4_Samples now...
Finished copying samples.
```

Samples will be installed to the directory ~/NVIDIA_CUDA-11.4_Samples.

2. \$ cd ~/NVIDIA_CUDA-11.4_Samples
\$ sudo make SMS=87 TARGET_ARCH=aarch64 TARGET_OS=linux TARGET_FS=\$NV_WORKSPACE/drive-
linux_src/filesystem/targetfs



Note: This will build the samples for the Orin GPU (SMS=87). To build for other GPUs, replace the SMS value with the target compute version.

2.7.2.2 How to Run the CUDA Samples

To run a CUDA sample application,

1. Copy the sample files of your choice to the target.
2. From the target, run the sample application.

For example:

```
$ cd ~/
$ rcp -r NVIDIA_CUDA-11.4_Samples <username>@<host>:/home/<username>
```

From the target:

```
$ cd ~/NVIDIA_CUDA-11.4_Samples/bin/aarch64/linux/release/
$ ./deviceQueryDrv
```

2.7.3 TensorRT

After you finish installing the TensorRT packages, the `/usr/src/tensorrt` folder is created on the development host.

For more information about:

- TensorRT Developer Guide and API Reference, see [NVIDIA TensorRT Documentation](#).
- How to cross-compile TensorRT samples, see [Sample Support Guide in NVIDIA TensorRT Documentation](#).

2.7.3.1 How to Build the TensorRT Samples

On the development host:

```
$ cd /usr/src/tensorrt/samples  
$ sudo make TARGET=aarch64
```

2.7.3.2 How to Run the TensorRT Samples on the Target

To run a TensorRT sample application,

1. Copy the sample files of your choice to the target.
2. From the target, run the sample application.

For example, from the host:

```
$ scp -r /usr/src/tensorrt <username>@<target ip address>:/home/nvidia/tensorrt
```

From the target:

```
$ cd /home/nvidia/tensorrt  
$ ./bin/sample_algorithm_selector
```

For further information, the TensorRT Developer guide and API reference documents are available at <https://docs.nvidia.com/deeplearning/tensorrt/index.html>, including sample cross-compile information at <https://docs.nvidia.com/deeplearning/tensorrt/sample-support-guide/index.html#cross-compiling-linux>.

2.8 DRIVE Platform Supported Boards

DRIVE OS supports the DRIVE Platform boards captured in the table below.

To flash your system, locate your **Part Number (P/N)** in the table below, and use the associated **Board Variant** as a parameter to flashing/binding utilities. If you don't know your **Part Number (P/N)**, please reference [How to Identify System Part Number \(P/N\)](#).

DRIVE Platform Part Numbers and Board Variants

Official Name	Platform	Part Number	Board Variant	DRIVE OS Version Required
Drive Orin N	p3898	900-63898-0000-100	p3898-a01	6.0.7.0+
		900-63898-0000-200	p3898-b00	6.0.8.0+
DRIVE Orin	p3663	940-63663-0000-100	p3663-a01	6.0.0.0+
		940-63663-0001-200	p3663-01-a02	6.0.3.0+
		940-63663-0002-200	p3663-02-a02	6.0.4.0+
DRIVE AGX Orin Developer Kit	p3710	<ul style="list-style-type: none"> > 940-63710-0010 TS1 > 940-63710-0010 TS2 > 940-63710-0010 A00 > 940-63710-0010 	<ul style="list-style-type: none"> > p3710-10-a01 > p3710-10-a01-ct02 	6.0.1.x+
		<ul style="list-style-type: none"> > 940-63710-0010 TS3 > 940-63710-0010 B00 	p3710-10-a03	6.0.2.1+
		<ul style="list-style-type: none"> > 940-63710-0010 TS4 > 940-63710-0010 C00 	p3710-10-a04	6.0.3.0+
		<ul style="list-style-type: none"> > 940-63710-0010 TS5 > 940-63710-0010 D00 	p3710-10-s05	6.0.3.1
		<ul style="list-style-type: none"> > 940-63710-0010 RC1 > 940-63710-0010 > 940-63710-0010 > 940-63710-0010 	p3710-10-s05	6.0.4.0
		<ul style="list-style-type: none"> > 940-63710-0012 TS3 > 940-63710-0012 B00 	p3710-12-a03	6.0.2.1+

Official Name	Platform	Part Number	Board Variant	DRIVE OS Version Required
		<ul style="list-style-type: none"> > 940-63710-0012 TS4 > 940-63710-0012 CO0 	p3710-12-a04	6.0.3.0+
		<ul style="list-style-type: none"> > 940-63710-0012 TS5 > 940-63710-0012 D00 	p3710-12-s05	6.0.3.1+
		<ul style="list-style-type: none"> > 940-63710-0012 RC1 > 940-63710-0012 > 694-63710-0012 D00 > 694-63710-0012 	p3710-12-s05	6.0.4.0

How to Identify System Part Number (P/N)

The **Part Number (P/N)** of your system is located on the bottom of the system. As an example, the picture below shows the bottom of the DRIVE AGX Orin Developer Kit.



Chapter 3. Setup and Configuration

The following sections describe how to set up and configure your system.

3.1 NVIDIA DRIVE AGX Board Setup

This section describes how to setup the hardware for the platform.

For board setup details, refer to the [NVIDIA DRIVE AGX Orin Developer Hardware Quick Start Guide](#).

3.2 Placing the Board In/Out of Recovery Mode

Recovery mode is used to load software updates, burn fuses, and run self-tests. To get into recovery mode:

To get into recovery mode, if the target has an AURIX terminal:

Command	Description
tegrarecovery [x1] [off on]	Set the device in recovery mode.
tegrareset [x1]	Reset the device. Default is X1.

Examples

To boot in recovery mode:

```
tegrarecovery x1 on  
tegrareset x1
```

To boot:

```
tegrarecovery x1 off  
tegrareset x1
```

3.3 Getting Started with DRIVE OS Linux 6.0 Development

This guide is for users of NVIDIA DRIVE® OS Linux running on the NVIDIA DRIVE AGX™ platform. After the initial setup between the target and the host is complete, you can build and run graphics, CUDA, and TensorRT sample applications. For more information on how to build and run the sample applications, see the Installation chapter.

3.3.1 Target SSH Access from the Linux Host

The target should have an IP address assigned automatically by the network or the host system to which it is connected. For more information about enabling the DHCP server on your development host to connect the target directly, see the NVIDIA DRIVE® OS 6.0 SDK Developer Guide section in [Setting Up Networking on the Host and Target](#).

During the initial setup of the target device, you are prompted to choose an SSH profile. If you choose the DRIVE OS Linux Secure Login profile, see the DRIVE OS 6.0 SDK Developer Guide section in [SSH Key-Based Authentication from Clients to Server](#) to create a key on the Linux host to connect to the target.

The DRIVE AGX Devkit is pre-flashed with the SSH server installed and enabled by default. If you flash with base DRIVE OS SDK Debian build, which uses the driveos-core-rfs filesystem, the SSH server will not be installed by default. Therefore, you need to install and set up by following the instructions as described in [DRIVE OS Linux SSH Server](#). If you install the driveos-oobe-desktop filesystem by following the instructions in the Finalize DRIVE AGX System Setup (Linux) section, the SSH server will be enabled by default.

To confirm the overall network and SSH installation, follow these steps:

1. Verify that the target can communicate to the host by using the ping command.

```
$ ping <target-IP>
```

2. Launch the SSH client.

```
$ ssh <user>@<target-IP>
```

Use the username and password that were set for the system setup during the initial boot process. Verify that same terms are used here. Follow the steps in the software guides found at [developer.nvidia.com/drive/start](#).

If the ping or ssh command is not successful and the target does not respond, you can launch the UART terminal (Minicom) to access the target.

- > Ensure that only one DRIVE AGX platform is connected to the USB port on the host system.
- > On the host system, verify that `ttyACM0` and `ttyACM1` are listed under the `/dev/` folder.
- > Run Minicom.

```
$ minicom -D /dev/ACM0
```

- > Check the IP address.

```
$ ifconfig eqos_0
```

3.4 Host/Target Setup and Configuration

This topic describes how to set up the networking between the host system and the target system, as well as target user account setup and environment for cross-compilation.

DRIVE OS Linux ships with an application that allows the end user to set up a user account using wizard prompts in the UART interface. When the target device boots for the first time after flashing, it automatically runs OEM-config.



Note:

- > OEM-config is not available in production SDK file systems including driveos-prod-rfs and driveos-prod-debug-rfs. If any of these file systems are flashed, you can log in by providing a username (nvidia) and password (nvidia).
- > To change user account information, such as username or password, or to add more user accounts, follow the steps described in the other sections of this topic to manage user accounts.

Before the target system boots for the first time, you must start a serial application on the host computer. For more instructions, check the [Terminal Emulation](#) section.

3.4.1 DRIVE OS Linux Configuration on the Target

Use the following steps to set up the NVIDIA DRIVE® OS Linux configuration on the target:

1. Allow the system to power on and boot up the operating system.

After the operating system boots, OEM-config will start and splash the user interface on the terminal with the welcome screen as shown below. If the OEM-config window does not appear, press the **Enter** key to refresh the screen, and the following prompt appears:

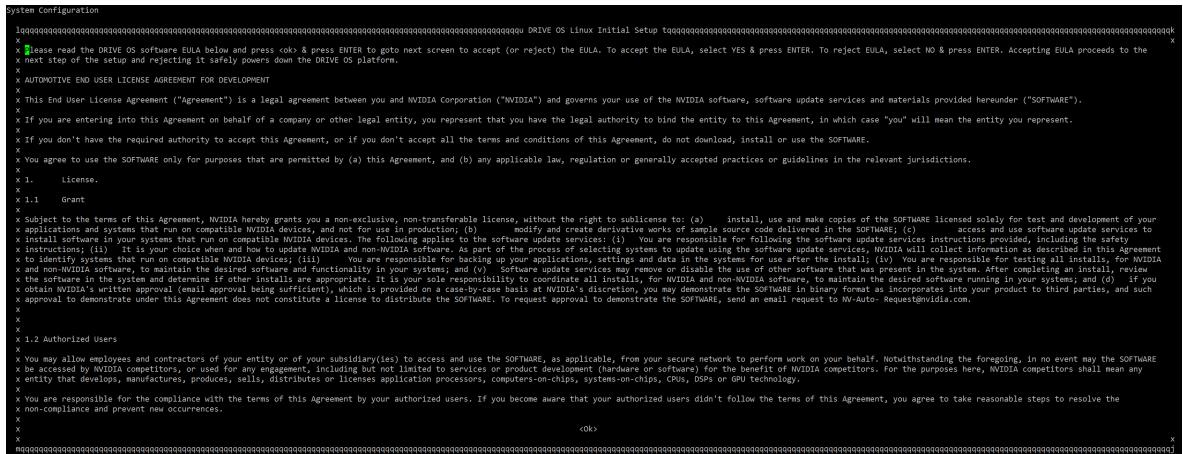
```

lqqqqu DRIVE OS Linux Initial Setup tqqqqqk
x
x Welcome to DRIVE OS SDK LINUX.          x
x                                         x
x Would you like to continue setup?    x
x                                         x
x      <Yes>           <No>        x
x                                         x
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj

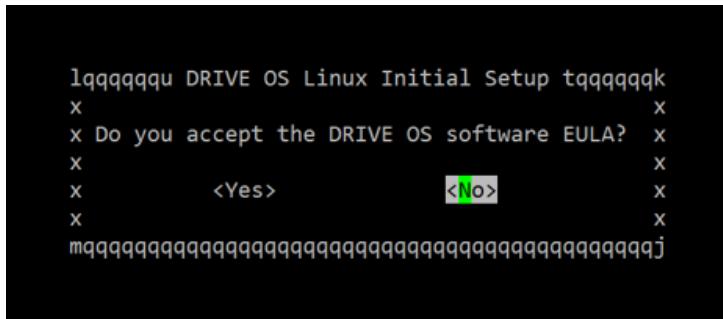
```

- > To proceed with OEM-config to display the EULA, use the **Tab** key to move to **<Yes>**, and then press **Enter**.
 - > To remain at the same screen, use the **Tab** key to move to **<No>**, and then press **Enter**.

2. After the EULA appears, use the **<PgUp>** and **<PgDn>** keys to scroll up and down when reading the EULA.



After reading the EULA, use the **Tab** key to move to **<Ok>**, and then press **Enter** to proceed to the acceptance of the EULA as follows:



- > To accept the EULA, use the **Tab** key to move to **<Yes>** and press **Enter** to proceed.
To proceed to the next step, accept the EULA.
 - > To reject the EULA, use the **Tab** key to move to **<No>**, and then press **Enter** to shut down the system safely.

3. Follow the OEM-config prompt to set up the first user in the system. The first user is always the administrator account (who can run the sudo commands).

The OEM-config will prompt you later to add more user accounts with the account type of non-administrator or administrator.

Press the **Enter** key to continue to set up the first user account.

4. Enter the full name of the first user, and then press **Enter** to continue.

5. Enter the first user's username.

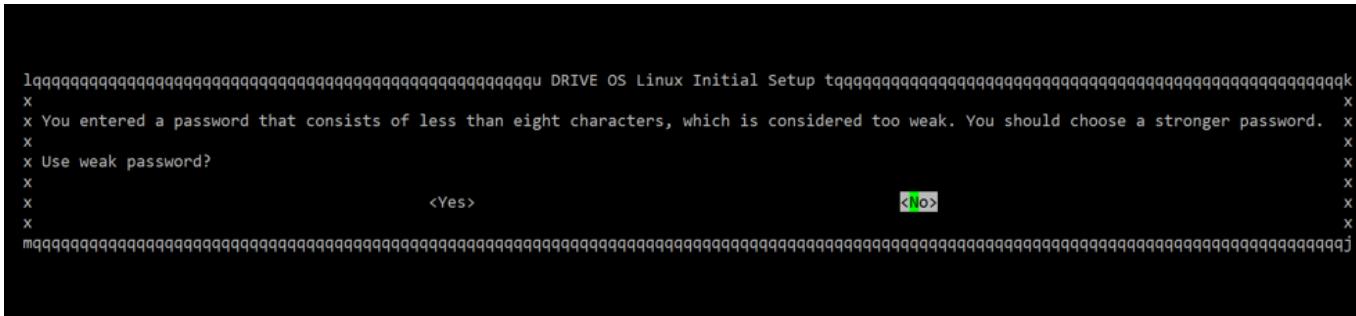
A valid username must satisfy this regular expression (regex) `^[a-zA-Z][a-zA-Z0-9_-]{0,31}$`, and OEM-config rejects invalid usernames.

The regex validates the username with the following conditions:

- > The username must not be empty.
 - > The username must start with either a lowercase letter or an underscore (_).
 - > The second and subsequent characters can be a lowercase letter, number, underscore (_), or hyphen (-).
 - > The username must be fewer than or equal to 32 characters in length.

6. Enter a password of your choice and re-enter the same password to confirm. The password must be made of ASCII characters.

7. After entering a password, use the **Tab** key to move to <Yes>, and then press **Enter** to proceed.

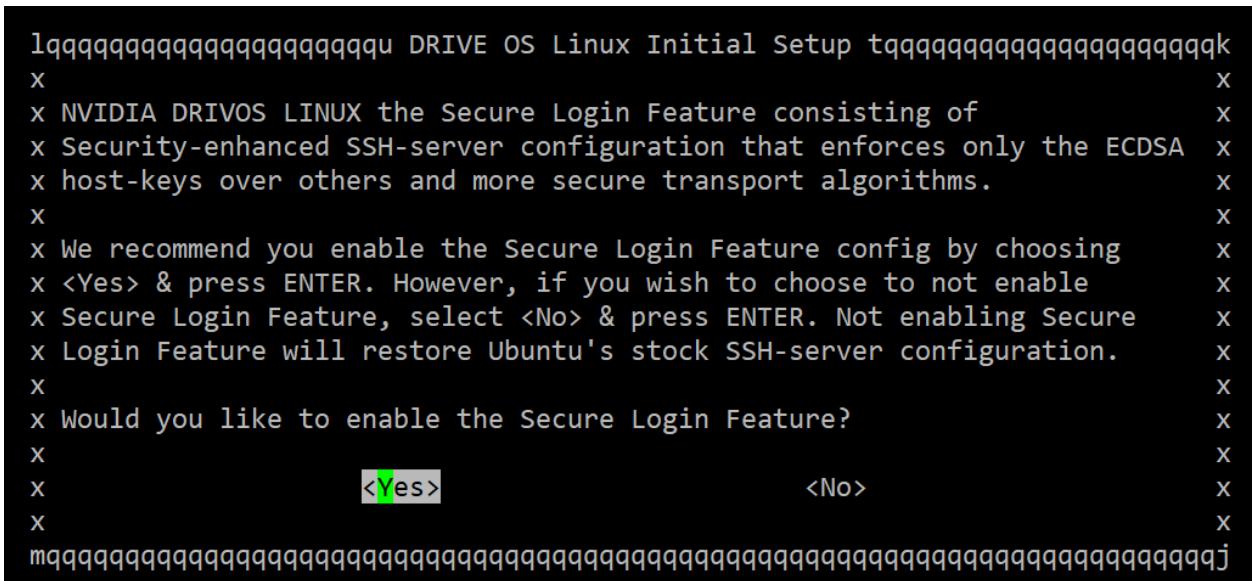


If the password created in step 6 is weak (for example, fewer than eight characters as shown in the preceding figure), the system recommends that you change the password with a warning message. To change the password, use the **Tab** key to move to **<No>**, and then press **Enter** to go back to step 6.

8. Follow the OEM-config prompt to enable or disable the secure login feature as shown in the following figure.

The secure login feature from NVIDIA DRIVE OS provides better security for SSH connections by using ECDSA-based algorithms. It is recommended that you enable the secure login feature.

- To enable the feature, use the **Tab** key to move to **<Yes>**, and then press **Enter**.
 - To disable the feature, use the **Tab** key to move to **<No>**, and then press **Enter**.



- Follow the OEM-config prompt to add more users.
 - To add more users, use the **Tab** key to move to **<Yes>**, and then press **Enter**.
 - Otherwise, use the **Tab** key to move to **<No>**, and then press **Enter**.

10.(Optional) If you choose to add more user accounts in step 9, OEM-config prompts you to choose the type of user for the new user account: admin or non-admin.

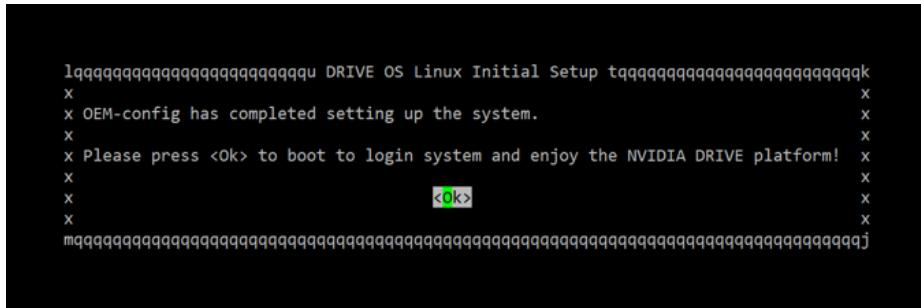
- To choose the administrator account, use the **Tab** key to move to **Admin**, and then press **Enter**.
 - To choose the non-administrator account, use the **Tab** key to move to **NonAdmin**, and then press **Enter**.

11.OEM-config setup is complete. Press **Enter** to exit the OEM-config screen to display the target command prompt.



Note: If OEM-config is interrupted by powering off the board or resetting the board on step 11 or before, your inputs are not recorded, and you must start over from step 1.

12. To enable OEM-config to provide information before executing steps in the backend to configure the system based on your inputs, press **Enter** to proceed.



3.4.2 DRIVE OS Linux User Management



Warning: Whenever any change is made to the filesystem such as adding/modifying/removing user login information, ensure that all data is saved prior to resetting. Execute a software shutdown command, such as halt, shutdown, or reboot to the target system to avoid data corruption; otherwise, file system corruption may occur. Once the target system is shut down, you may use physical/electrical shutdown or reset commands, such as tegrareset or aurixreset in the AURIX command terminal.

NVIDIA DRIVE OS Linux uses standard Ubuntu tools to manage user accounts. Only admin-users (such as sudoers or root user) can change user accounts. The following steps assume that you are an admin user. Enter the password of your current user.

3.4.2.1 Steps to Change the Username and Password

The following steps describe how to change your username and password in the DRIVE OS Linux filesystem.



Warning: Whenever any change is made to the filesystem such as adding/modifying/removing user login information, ensure that all data is saved prior to resetting. Execute a software shutdown command, such as halt, shutdown, or reboot to the target system to avoid data corruption; otherwise, file system corruption may occur. Once the target system is shut down, you may use physical/electrical shutdown or reset commands, such as tegrareset or aurixreset in the AURIX command terminal.

Changing the Username

1. Enable administrative account.
 - > Enable the root administrative account by setting a password for the root account. Use the following command to set a password for root account. You will be prompted for the password of your current user.


```
$ sudo passwd
```
2. Close existing user sessions.

- > Log out of all sessions, including GUI and consoles of the user that you are changing the username for. You cannot change the username if a session is still alive.

- > Use the following command to exit console sessions:

```
$ exit
```

3. Log into the system as root account.

- > After completing step #2, you will be presented with a login prompt. Proceed to log in using the credentials you established for the root account.

4. Change the username.

- > Change the username of the user from \${USERNAME1} to \${USERNAME2} with the following commands:

```
$ usermod -m -d /home/${USERNAME2} -l ${USERNAME2} ${USERNAME1}
$ groupmod -n ${USERNAME2} ${USERNAME1}
```

- > Log out of the root account using the following command:

```
$ exit
```

You have successfully changed the username. Proceed to log in with the new username.

Changing the Password



Note: When changing the password for the current user, enter the command without sudo as listed below. Entering sudo passwd changes the password for the root user.

1. Enter the following command to update password.

```
$ passwd
```

Enter your existing password and enter the new password.

```
Changing password for <user>.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully.
```

2. Enter the current password and then the new password.

3.4.2.2 Adding a Non-Admin User

1. Execute adduser with the new username as input and fill in the information when prompted.

```
$ sudo adduser <user>
```



Note:

Non-Admin users can be added to specific groups to get privileged access to components that are otherwise accessible only if the user is root. See [Common Groups used in DRIVE OS Linux Filesystems](#).

Ensure the Non-Admin user is not added to the 'sudo' or 'adm' group, as this will allow them to run any commands using sudo.

2. Execute usermod to add the user to the required groups as per the use case requirements of the user.

```
$ sudo usermod -aG <groups> <user>
```

3.4.2.2.1 Example

```
$ sudo adduser test
Adding user `test' ...
Adding new group `test' (1001) ...
Adding new user `test' (1001) with group `test' ...
Creating home directory `/home/test' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for test
Enter the new value, or press ENTER for the default
      Full Name []:
      Room Number []:
      Work Phone []:
      Home Phone []:
      Other []
Is the information correct? [Y/n] Y
```

3.4.2.3 Adding an Admin User

1. Create a non-admin user <user> by following the instructions in the section [Adding a Non-Admin User](#).
2. Make the created user <user> an admin user by adding the user to all the groups specified in '[Common Groups used in DRIVE OS Linux Filesystems](#)' following the instructions in the section [Adding a Non-Admin User](#) above.

3.4.2.4 Common Groups Used in DRIVE OS Linux Filesystems

Group	Description
adm	Group adm is used for system monitoring tasks. Members of this group can read many log files in /var/log.
audio	This group can be used to give a set of users, access to sound devices. (e.g., sound timers)
dialout	Members of this group gets full and direct access to serial ports.
plugdev	Allows members to mount (only with the options nodev and nosuid, for security reasons) and umount removable devices through pmount.

Group	Description
sudo	Members of this group can execute any command with sudo (/etc/sudoers)
video	This group can be used to give a set of users access to a video device (like the framebuffer, the videocard or a webcam). Users may require to be added to this group to run DRIVE OS graphics samples.
debug	This group can be used to give a set of users, access to profile and debug data for GPUs in the system.

3.4.2.4.1 Removing a User

The steps to remove an admin or non-admin users are identical. Removing users using the steps below also removes their /home/<user> directory.

To remove a user, enter:

```
$ sudo deluser --backup --remove-home <user>
```

This backs up and creates a tarball (<user>.tar.bz2) of the deleted user data in the current working directory.

3.4.2.4.2 Setting Password as an Admin User

As admin-users are sudoers, they can set the password of any user account with username <username> using the command below. Enter the password of your current user.

```
$ sudo passwd <username>
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully.
```

3.4.2.4.3 Updating a Password as a Non-Admin User



Note: When changing the password for the current user, enter the command without sudo as listed below. Entering sudo passwd changes the password for the root user.

Non-admin users can only update their own passwords. The steps below update your user account's password:

1. Enter the following command:

```
$ passwd
```

2. Enter your existing password and enter the new password.

```
Changing password for <user>.
(current) UNIX password:
Enter new UNIX password:
```

```
Retype new UNIX password:
passwd: password updated successfully.
```

3. Enter the current password and then the new password.

3.4.2.4.4 Disabling a User Account

The following step allows an admin user to disable the user account with a username \$USER:

```
$ sudo usermod --expiredate 1 $USER
```

Disabling the user account ensures that the account is unusable in any context in the filesystem. The disabled user (\$USER) can be re-enabled with the following command:

```
$ sudo usermod --expiredate "" $USER
```

3.4.2.5 Disabling the Secure Login Feature

Disabling Secure Login Feature (SLF) removes NVIDIA DRIVE OS security SSH server config and replaces it with stock ubuntu SSH server config. There are two (2) options to disable the secure login feature:

1. If the user wants to disable the SLF, the first option is available during the target setup process using OEM-config and choosing No for the following prompt:
Would you like to enable or disable the Secure Login Feature?
2. The second option is to manually copy the ubuntu stock SSH server config to the system's SSH server config:
 - a. `sudo cp /usr/share/openssh/sshd_config /etc/ssh/sshd_config`
 - b. `sudo systemctl restart ssh`

If the user wants to re-enable SLF after the steps above, copy the NVIDIA DRIVE OS security SSH server config to the system and restart SSH server.

1. `sudo cp /etc/nvidia/configs/sshd_config.driveos /etc/ssh/sshd_config`
2. `sudo systemctl restart ssh`

3.4.3 DRIVE OS Linux SSH Server

The shipped NVIDIA DRIVE OS LINUX filesystem driveos-oobe-rfs contains SSH server. However, driveos-core-rfs does not include SSH server. Use the steps below to install SSH server.

3.4.3.1 Install/Update SSH Server

1. Ensure platform is connected to the internet
2. Update the existing apt database:

```
# apt-get update
```

3. Install SSH server package:

```
# apt-get install ssh
```

- When prompted by apt-get (see prompt below), choose the option keep the local version currently installed to ensure DRIVE OS SSH configuration is applied.

3.4.3.2 Setting Up SSH Server Service

After SSH server is available in the filesystem, use the steps below as root user to start and configure SSH server service to run on every boot:

- Remove the stamp file to unblock SSH server:

```
$ sudo rm -f /etc/ssh/sshd_not_to_be_run
```

- Start SSH server service on the current boot:

```
$ sudo systemctl start ssh
```

- Start service to add SSH host-keys to the target:

```
$ sudo systemctl start nv_ssh_host_keys
```

After completing the steps above, the SSH server service is started. Additionally, it runs on every boot. SSH clients may now connect to this SSH server.

3.4.3.3 SSH Server Configuration File

NVIDIA DRIVE OS Linux contains an SSH server configuration file [sshd_config](#) that pre-configures the following :

- > Permits only connections with Elliptic Curve Digital Signature Algorithm (ECDSA) host key.
- > Disables compression.
- > Permits specific key exchange, host signature, and session encryption algorithms.

NVIDIA DRIVE OS Linux recommends that you configure strong security options for the following SSH parameters:

- > Client Verification via key exchange: Parameter KexAlgorithms from [sshd_config](#) to set the supported key type(s) for client verification.
- > Host Signature: Parameter HostbasedAcceptedKeyTypes from [sshd_config](#) to set the supported key type(s) for host verification.
- > Session Encryption: Parameter Ciphers from [sshd_config](#) to set the supported SSH session encryption algorithm(s).

The table below suggests recommended strong values to enable recommended security features; the default option is highlighted in the table below.:

Option	Client Verification	Host Signature	Session Encryption
1	ecdh-sha2-nistp256 ECDH [SP 800-56A] over secp256r1 [SEC2-V2] with SHA-256 [FIPS 180-4]	ecdsa-sha2-nistp256-cert-v01@openssh.com ECDSA [FIPS 186-4][ANS X9.62] over secp256r1 [SEC2-V2] with SHA-256 [FIPS 180-4]	aes128-gcm@openssh.com GCM [SP 800-38D] with AES-128 [FIPS 197]
2	ecdh-sha2-nistp384 ECDH [SP 800-56A] over secp384r1 [SEC2-V2] with SHA-384 [FIPS 180-4]	ecdsa-sha2-nistp384-cert-v01@openssh.com ECDSA [FIPS 186-4][ANS X9.62] over secp384r1 [SEC2-V2] with SHA-384 [FIPS 180-4]	aes256-gcm@openssh.com GCM [SP 800-38D] with AES-256 [FIPS 197]
3	(Default) ecdh-sha2-nistp521 ECDH [SP 800-56A] over secp521r1 [SEC2-V2] with SHA-512 [FIPS 180-4]	(Default) ecdsa-sha2-nistp521-cert-v01@openssh.com ECDSA [FIPS 186-4][ANS X9.62] over secp521r1 [SEC2-V2] with SHA-512 [FIPS 180-4]	(Default) aes256-gcm@openssh.com GCM [SP 800-38D] with AES-256 [FIPS 197]

3.4.3.4 Switching Options in the SSHD Configuration File

DRIVE OS Linux contains the snippets for all three (3) options with exactly one option (option 3) enabled as shown below. The options 1, 2, and 3 are mutually exclusive and exactly only one option can be active at a time.

```
# option 1
# Use algorithms as per DRIVE recommended options: 1,2, or 3
#KexAlgorithms ecdh-sha2-nistp256
#HostbasedAcceptedKeyTypes ecdsa-sha2-nistp256-cert-v01@openssh.com
#Ciphers aes128-gcm@openssh.com
# option 2
#KexAlgorithms ecdh-sha2-nistp384
#HostbasedAcceptedKeyTypes ecdsa-sha2-nistp384-cert-v01@openssh.com
#Ciphers aes256-gcm@openssh.com
# option 3 (default)
KexAlgorithms ecdh-sha2-nistp521
HostbasedAcceptedKeyTypes ecdsa-sha2-nistp521-cert-v01@openssh.com
Ciphers aes256-gcm@openssh.com
```

To switch from one option to another

1. Comment out the three (3) lines corresponding to the current option.
2. Uncomment the three (3) lines corresponding to the new option.

3. Restart SSHD using the command below as the root user:

```
$ sudo systemctl restart ssh
```

As an example, assume that the current option is option 3 (from the snippet above).

To switch to option 2, execute the following steps:

1. Comment out the three (3) lines following # option 3.
2. Uncomment the three (3) lines from the line below # option 2 until the line above # option 3.
3. Restart SSHD using the command below as the root user:

```
$ sudo systemctl restart ssh
```

4. SSHD now uses encryption algorithms from option 2.

3.4.3.5 SSH Key-Based Authentication from Clients to Server

DRIVE OS Linux shipped SSHD tries key-based authentication first and falls back to password-based authentication when the former is unavailable. The following sections describe how to set up a user-specific authentication key, add it to the list of trusted-keys, and use the key to authenticate (instead of the password) to the SSH server.

3.4.3.5.1 Create a new key-pair at the client side

The first step is to create a public/private key pair using the command:

```
$ ssh-keygen -t ecdsa
```

Continue through the prompts by pressing ENTER and these steps save the key to `~/.ssh`.

3.4.3.5.2 Set up the server side to register key

The next step is to add the contents of the client (user-specific authentication) public key `~/.ssh/id_ecdsa.pub` into the text file `~/.ssh/authorized_keys`.

3.4.3.5.3 Use the new key at the client side to connect to the server

Only the user who created the key can log in to the SSHD server (using the registered keys). At the client side, connect to the server using the `ssh` command:

```
$ ssh <user>@<target_ip>
```

The setup done above is one-directional (i.e., from client to server). To similarly setup in a reverse direction (i.e., from server to client), the above steps must be swapped with regards to client/server and executed to setup key-based authentication from the server to client-side.

3.4.4 Enabling VNC Remote Access

To install/run `x11vnc`:

```
$ sudo apt update
```

```
$ sudo apt install x11vnc
```

VNC command when running X server manually:

```
$ sudo x11vnc -display :0 -noxrecord -noxfixes -noxdamage -forever -loop -nopw
```

VNC command when running full desktop:

1. Configure the target desktop for auto login:

```
$ sudo vi /etc/gdm3/custom.conf
```

2. Uncomment/edit the following lines as shown below:

```
# Enabling automatic login
AutomaticLoginEnable = true
AutomaticLogin = <user>
```

3. Save/exit the file and restart the gdm service:

```
$ sudo systemctl restart gdm3.service
```

4. Start the VNC server:

```
$ sudo x11vnc -display :0 -noxrecord -noxfixes -noxdamage -forever -loop -nopw -
auth /run/user/1000/gdm/Xauthority
```

3.4.5 Setting Up Networking on the Host and Target

Use the information in this topic to set up the network between an Ubuntu host and your NVIDIA DRIVE® platform. The steps to set up networking includes configuring the DHCP and NFS servers, configuring networking on the host computer, etc.

3.4.5.1 Configuring the Network Interface

Configuring the network interface for your device requires:

- > Connecting the board to the host Linux development system.
- > Configuring the network interface.



Warning: To avoid auto detection conflicts with the adapter, do not configure these items with the network manager.

3.4.5.1.1 Connecting the Target to the Host Using the Network Interface

You can connect your target board to the Linux Ubuntu host machine using a private LAN. This private LAN is not the LAN connecting the host to the Internet. Connect through Onboard 1Gb Ethernet connector.

3.4.5.1.1.1 To use onboard HMTD connector to connect the board and host on the private LAN

1. To use onboard 1Gb ethernet, connect to the DRIVE platform 1 Gb RJ45 Ethernet port on the target as described in the [DRIVE AGX Orin Developer Kit Hardware Quick Start Guide](#).
2. Obtain a CAT-6 crossover patch cable and plug one of the RJ45 male ends into the RJ45 port of target from step 1.

3. Plug the other male end of the CAT-6 crossover patch cable into the RJ45 port of HOST machine or to RJ45 port of USB-to-Ethernet adapter (e.g., Dlink DUB-E100 (smsc95xx)) connected on your host machine.

3.4.5.1.2 Configuring the Private LAN to the Target Network

Use the following procedure to configure the host interface for the private LAN connected to the target platform. The procedure assumes eth1 is the Ethernet port on the host PC connected to the NVIDIA board.

3.4.5.1.2.1 To configure the private LAN to the target

1. Determine which host eth<n> port is connected to the *target*, where <n> is the port instance.
 - > Find the eth device with the following command:
`dmesg | grep -i eth`
 - > In the grep results, identify the eth<n> port for the Dlink DUB-E100 (smsc95xx) or similar USB Ethernet adapter.
 - > For example, the following dmesg result indicates that the eth1 port is connected to the target:

```
[1310932.166153] smsc95xx 2-5.1:1.0: eth1: register
'smsc95xx' at usb-0000:00:1d.7-5.1, smsc95xx USB 2.0
Ethernet, 00:04:4b:1b:32:6b
```

2. On the host, find and edit the following file:

```
/etc/network/interfaces
```

This file is read-only, so you must open it with administrative privileges, for example:

```
sudo vim /etc/network/interfaces
```

3. Depending on your connection at the host, modify the interfaces file:

- > Additional NIC card/adapter: Add the following to the interfaces file:

```
auto eth1
iface eth1 inet static
    address 10.0.0.1
    netmask 255.255.255.0
```

- > USB Ethernet adapter: Add the following to the interfaces file:

```
auto eth1
allow-hotplug eth1
iface eth1 inet static
    address 10.0.0.1
    netmask 255.255.255.0
    post-up /etc/init.d/isc-dhcp-server restart
```

4. Restart the host's networking with the following command:

```
sudo /etc/init.d/networking restart
```

5. Hard reboot the host system.

3.4.5.2 Configuring the DHCP and NFS Server on the Host

The DHCP server on the host is used to assign the IP address to the target system and the NFS server is used to mount the root file system on the NVIDIA target system using NFS.

If the DHCP and NFS servers are not yet installed on the host, the installer installs and configures them. Alternatively, those servers can be installed and configured as follows.

3.4.5.2.1 To set the DHCP server

1. Install the DHCP server on the host:

```
sudo apt-get install isc-dhcp-server
```

2. Specify the interface on which the server should listen for leasing an IP address from the target over the private LAN.

- > Locate and edit the following file:

```
/etc/default/isc-dhcp-server
```

This file is read-only, so you must open it with administrator privileges.

- > Modify the `isc-dhcp-server` file to set `INTERFACES` to the `eth<n>` connection you determined when connecting your network interface.

For example, add the following line if the DHCP server should listen on `eth1` interface.

```
INTERFACES="eth1"
```

Changing the interface to "eth3" can result in the following error:

```
udevd[148]: error changing net interface name eth0 to eth3:  
Device or resource busy
```

To resolve this error, delete the `/etc/udev/rules.d/70-persistent-net.rules` file.

3. Configure your host DHCP server for the target interface.

- > Locate and edit the following file:

```
/etc/dhcp/dhcpd.conf
```

Because the file is read-only, open it with administrator privileges.

- > Modify the `dhcpd.conf` file to contain the following:

```
ddns-update-style none;  
allow bootp;  
subnet 10.0.0.0 netmask 255.255.255.0 {  
    option routers 10.0.0.1;  
    option domain-name <domain_name>;  
    option domain-name-servers <DNS1>, <DNS2>, ... ;  
    default-lease-time 345600;  
    max-lease-time 31557600;  
    range 10.0.0.2 10.0.0.254;  
    option root-path "10.0.0.1:<top>/drive-linux/  
    /filesystem/targetfs, wsize=8192, rsize=8192, v3";
```

```
}
```

Where:

- <domain_name> is your company domain name.
- <DNS1>, <DNS2> are the DNS servers that you already added to the /etc/resolv.conf file on your host system. Multiple DNS servers are separated by commas. For example, the Google public DNS IP addresses are formatted as:

```
8.8.8.8, 8.8.4.4
```

4. Restart the DHCP server:

```
sudo /etc/init.d/isc-dhcp-server restart
```

3.4.5.2.2 To set the NFS server

1. Install the NFS server on the host using apt-get:

```
sudo apt-get install nfs-kernel-server nfs-common portmap
```

2. Locate and edit the following file:

```
/etc/exports
```

Add the corresponding path to the *target* file system:

```
<top>/drive-linux *(async,rw,no_root_squash,no_all_squash,no_subtree_check)
```

This change exports the target file system.

3. Restart the NFS server:

```
sudo /etc/init.d/nfs-kernel-server restart
sudo exportfs -a
```

3.4.5.2.3 To enable Internet access from the target

On the Linux host, enter these commands to enable settings on the host:

```
$ sudo sysctl -w net.ipv4.ip_forward=1
$ sudo iptables -F
$ sudo iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

Where eth1 is the interface connected to the network that is connected to the Internet on the host.

```
$ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Where eth0 is the private LAN interface connected to target from the host.

3.4.5.3 Configuring Static IP to the Device

The following procedure describes how to assign a static IP to DUT's 1G or 10G interface for Ubuntu 17.10, 18.04, and higher.

1. Create the .yaml configuration file for netplan in /etc/netplan if it does not already exist. A number prefix is used for orderly processing. For more information, see `man netplan`.



Note: Replace eth0 with the interface name of your system.

```
sudo cat >/etc/netplan/01-network-manager-all.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0: # Use your interface name
      dhcp4: no
      dhcp6: no
      addresses:
        - 10.0.0.X/22    # Your IP address
      gateway4: 10.0.0.1      # Your gateway
      nameservers:
        search: [nvidia.com, client.nvidia.com,
nvclient.nvidia.com] #Your corp domain names
      addresses: [DNS1,DNS2]
```

2. After the editing of the 01-network-manager-all.yaml file is complete, apply the settings using the following command.

```
sudo netplan apply
```

3. Review the configuration.

```
ip addr
ip show
```

3.5 Camera Setup and Configuration

The platform provides multiple video and camera ports.

Before using these cameras with the NvMedia sample applications, the cameras must be attached to the ports in a specific order. If you fail to do so, NvMedia reports errors.



Warning:

Before connecting/disconnecting cameras to/from the platform, disconnect the power. Failure to do so may damage the camera.

For information on setting up the board, refer [DRIVE AGX Orin Developer Kit_Hardware_Quick Start Guide](#).

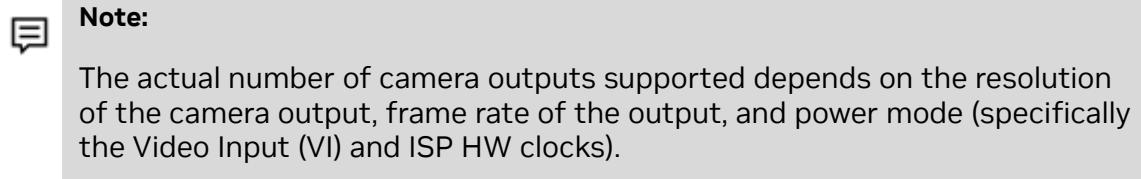
For information on the bandwidth limitation of camera ports, refer to [Camera Commands](#).

3.5.1 Camera Interfaces

Maxim Integrated GMSL SERDES

The NVIDIA DRIVE® AGX Orin platform (P3710) provides GMSL camera interfaces. The GMSL camera interfaces:

- > Provide 16 simultaneous GMSL camera inputs.



- > Route camera data to either SoC.
- > SoC generates a TSC_EDGE_OUT signal on a GPIO pin that is connected to all deserializers. This common PWM signal is then forwarded to all 16 GMSL cameras to achieve frame synchronization.

For information on the connectors, see the rear view board image in [Setting Up Your Platform](#).

TI FPD-Link SERDES Support

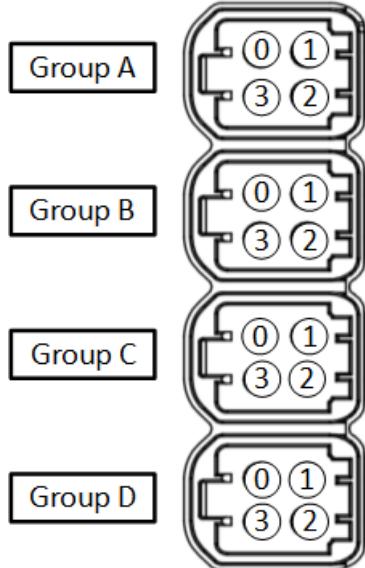
Initial support for TI FPD-Link IV SERDES links is available in Drive OS 6.0.6.0, with the driver and source for the DS90UB9724 deserializer provided in conjunction with the DS90UB971 serializer. This is available in SIPL, with the P3714-B00 FPDLINK CIM. Driver and configuration support for the IMX728 FPD-Link camera module using the DS90UB971 serializer is also available.

3.5.2 Mapping Connectors

Each GMSL camera group can be routed to the chip using a GMSL deserializer.

- > GMSL cameras are organized into quads. For example, A, B, C and D as available.
- > Different cameras may be used in different quads.
- > When connecting or disconnecting the GMSL cameras, ensure the power is off. If the power is ON, the GMSL camera or the platform may be damaged.
- > For Camera group A and B, I2C bus speed is 1Mhz. For Camera group C and D, I2C bus speed is 400KHz. Connect the GMSL cameras to the group with the correct I2C bus speed.

The camera mapping for the DRIVE AGX Orin Developer Kit (P3710-10) board is as follows:



3.5.3 Connecting the Cameras



Warning:

The GMSL camera must be 8V tolerant. Refer to the platform datasheet for details on the electrical requirements for the GMSL camera.

Always turn off main power before connecting or disconnecting cameras.

To connect multiple cameras using the quad camera breakout cable

1. Connect the quad camera breakout cable to the platform camera group A.
2. Using the Fakra coax cable, connect the GMSL camera to the other end of the quad camera breakout cable.
3. Connect any subsequent cameras to each connector of the quad camera breakout cable.
4. The mapping of the quad camera breakout cable:
 - > Green: A0
 - > Red: A1
 - > Blue: A2
 - > White: A3

To run the sample applications

1. Start camera capture using commands.

For example, for Valeo IMX728 B1 camera module connected to A0 port in the Group A, and an HDMI monitor is connected to the DP (DisplayPort) port. The commands are below.

Before using the -d option with the nvsipl_camera application, you must complete the following requirements:

- a. Terminate Ubuntu Desktop
 - > Terminate Desktop:

```
systemctl disable gdm
systemctl stop gdm
```
 - > Alternatively:

```
sudo service lightdm stop
sudo service gdm stop
```
 - b. Terminate X11

```
sudo pkill Xorg
```
 - c. Terminate any active Vulkan or OpenGL or Graphics applications.
 - d. `sudo lsmod | grep nvidia-drm`
 - e. If the output shows nvidia-drm, uninstall drm:

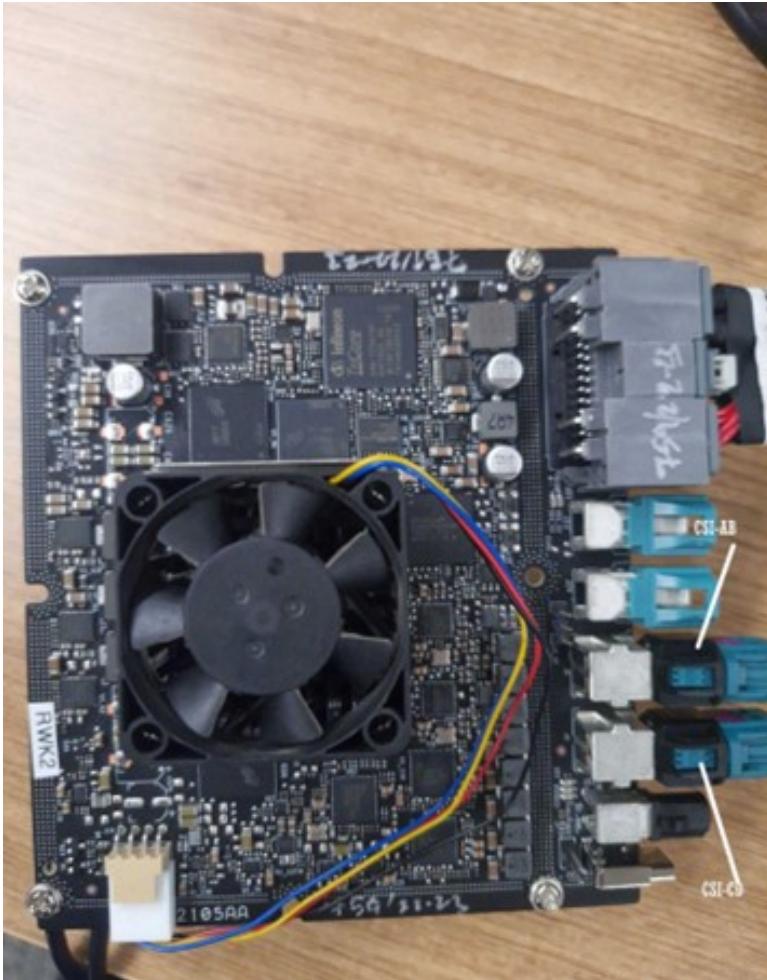
```
sudo rmmod nvidia-drm
```
2. Launch a terminal window and navigate to the following directory.
 - > On Ubuntu rootfs:
`/opt/nvidia/drive-linux/samples/nvmedia/nvsipl/test/camera`
 - > On Yocto rootfs:
`/opt/root/samples/nvmedia/nvsipl/test/camera`
 3. Enter this command.

```
sudo ./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4" --link-enable-masks
"0x0001 0x0000 0x0000 0x0000" -d 0
```

Where:

- > V728S1-120V1-FWC_CPHY_x4 specifies the name of the platform configuration that describes the connection of image sensors to Orin based platforms.
 - > -d 1 specifies the display number.
 - > -w 1 specifies display window ID.
4. To obtain the available display devices for Tegra, execute the following command:
`./nvsipl_camera -h`
The available display devices are identified.
 5. Select the desired display ID.

3.5.4 Setting up Cameras on the P3898 Platform



Note: The P3898 platform has connections for two camera groups. There are two different deserializers used per camera group, and each one supports the different link speed between the serializer and the deserializer. Connect the camera module to the compatible deserializer. Otherwise, the link-lock error is reported.

P3898

Camera Group A : MAX96724 (3 or 6Gbps)

Camera Group B : MAX96724F (3 Gbps)

3.6 Supported Sensors

The following sensors are supported:

Camera Make / Model	Supported Release
Sony IMX728 8MP 120FOV Camera with FPDLINK Serializer	6.0.6
Valeo IMX728 B3	6.0.3
OmniVision OV2311 2MP GS-IR DMS 55	6.0.3
Entron F008A120RMOAES	6.0.2
Valeo IMX728 B2	6.0.2
Valeo IMX728 B1	6.0.1

For more information, refer to [DRIVE Hyperion 8.1 Sensors and Accessories](#) and [DRIVE AGX Orin Sensors and Accessories](#).

3.7 MCU Setup and Usage

MCU Device Types

On the NVIDIA DRIVE AGX Orin™ Platform, the Safety MCU provided is the Infineon AURIX TC397X B-Step. The device is supported with Vector firmware.

Accessing the SoC from UART

Before using UART on the host system:

- > Install Foundation package.
- > Install the following packages on the host system:

```
sudo apt-get install p7zip-full
sudo apt-get install dpkg-dev
```

3.7.1 Software Setup on the Linux Host



Note: Connect the USB cable from USB 2 Port on NVIDIA DRIVE Orin™ to the host system.

Setting Up and Configuring Minicom

For detailed information, see using [Minicom](#).

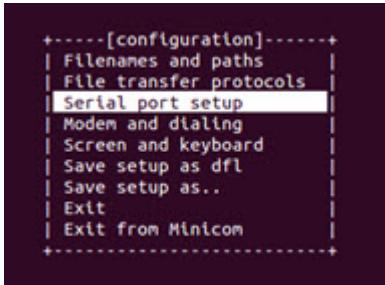
1. If Minicom is not installed, install it with the command.

```
sudo apt-get install minicom
```

2. Configure Minicom on the host as follows.

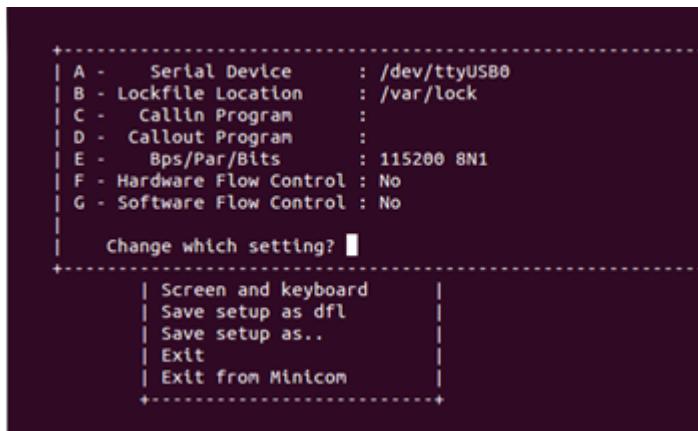
```
sudo minicom -s
```

- From the configuration dialog, select **Serial port setup**.



- Define the configuration as follows:

- > Serial Device: /dev/ttyUSB0
- > Lockfile location: /var/lock
- > Bps/Par/Bits: 115200 8N1
- > Hardware Flow Control: No
- > Software Flow Control: No



- Save the setup as the default setup.
- Launch Minicom for a specific device using the -D option.

- > For P3663

```
sudo minicom -D /dev/ttyUSB1
• ttyUSB0 for NVIDIA DRIVE Orin™ X1
• ttyUSB1 for AURIX
```



Note: The USB[X] will be assigned based on which is plugged in first to the host.
The previous configuration is just an example.

- > For P3710

```
sudo minicom -D /dev/ttyACM1
• ttyACM0 for NVIDIA DRIVE Orin™ X1
• ttyACM1 for AURIX
```

- Query the relevant serial devices on the host system with the command.

- > For P3663

```
dmesg | grep ttyUSB
```

- > For P3710

```
dmesg | grep ttyACM
```

3.7.2 Using the MCU Console

After the target is connected, the MCU prompt displays in the serial console. Use the MCU console to run commands and get information from the MCU.

UART Settings

The serial communication settings for the MCU console are as follows:

- > Baud rate: 115,200 bits per second (bps)
- > Data: 8-bit, no parity, 1 stop

MCU Console Commands

The MCU console commands are as follows.

Command	Description
aurixreset	Power cycle board including Aurix.
date [0x12345678]	Set date = 0x12345678 on Aurix in seconds(Unix style).
version	Show the current flashed/running firmware version.
cansetbr [a b c d e f] [125 250 500]	Switch baudrate of requested CAN channel to supported baudrate.
cycliccanon [CAN CANFD CANEx All]	Switch on Cyclic CAN/CANFD/CANEx messages (for AURIX).
cycliccanoff [CAN CANFD CANEx All]	Switch off Cyclic CAN/CANFD/CANEx messages (for AURIX).
i2cread x y z n	Read 'n' bytes from device 'y' at offset 'z' of I2C-controller 'x' (all hex-byte-values).
i2cscan [0]	Scan all connected I2C devices at I2C module 0.
i2cwrite x y z n b0 b1 ...	Write 'n' bytes to device 'y' at offset 'z', values Byte-b0, Byte-b1 of I2C-controller 'x' (all hex-byte-values).
inforomdump	Dump the Inforom content in hex format.

<code>help</code>	Show all available commands supported.
<code>poweron</code>	Turn main power on and release Tegra resets.
<code>poweroff [safeshutdown]</code>	Set the outputs to system power off state.
<code>readNvRam</code>	Read NvRam content.
<code>setdfltbtchain [x1] [A B C D]</code>	Set default boot chain configuration for Tegra x1 to A B C D.
<code>getdfltbtchain [x1]</code>	Get default boot chain configuration for Tegra x1.
<code>setnxtbtchain [x1] [A B C D]</code>	Set next boot chain for Tegra x1 to A B C D.
<code>getnxtbtchain [x1]</code>	Get next boot chain configuration for x1.
<code>tegrarecovery [x1] [on off]</code>	Set Tegra X1 in recovery mode.
<code>tegrareset [x1] [h]</code>	Reset Tegra X1. Default is 'X1'. If 'h' option is supplied, then hold the specified Tegra in reset.
<code>gptpon</code>	Enable Aurix as PTP master.
<code>gptpoff</code>	Disable Aurix as PTP master.
<code>status</code>	Display the Aurix internal status.
<code>setnetworkconfig [0-10]</code>	Set network configuration.
<code>getnetworkconfig</code>	Get network Configuration.
<code>show_fanrpm</code>	Show RPM of all fans.
<code>set_fanpwm [1 2] [DutyCyclePercentage]</code>	Set Fan PWM, that is, duty cycle.
<code>get_temperature</code>	Get temperature.
<code>send_temperature [Temperature]</code>	Send temperature.
<code>set_tmonerror [init rdwr lcltg1 rmttg1 tmonalert tmontherm]</code>	Inject false case.
<code>setistconfig [config]</code>	Set Tegra IST configuration.
<code>istabort</code>	Abort IST.
<code>getistinfo</code>	Get Tegra IST information.
<code>getistconfig</code>	Get Tegra IST configuration.
<code>getistresult [config]</code>	Get Tegra IST results.

<code>entersc7</code>	Enter SC7 power mode.
<code>exitsc7</code>	Exit from SC7 power mode.
<code>readvrs10status</code>	Read VRS10 status register.
<code>readvrs11status</code>	Read VRS11 status register.
<code>vrs10injecterror [inputs regwrite nirqtoggle corruptfint]</code>	Inject faults in VRS10.
<code>vrs11injecterror [regwrite 1 regwrite 2 nirqtoggle 1 nirqtoggle 2 corruptfint 1 corruptfint 2]</code>	Inject faults in VRS11.
<code>vrs12injecterror [overror uverror wrongcrc plauscheck]</code>	Inject faults in VRS12.

3.7.3 SoC to Microcontroller Communications

Common Interface for SoC to MCU Communication

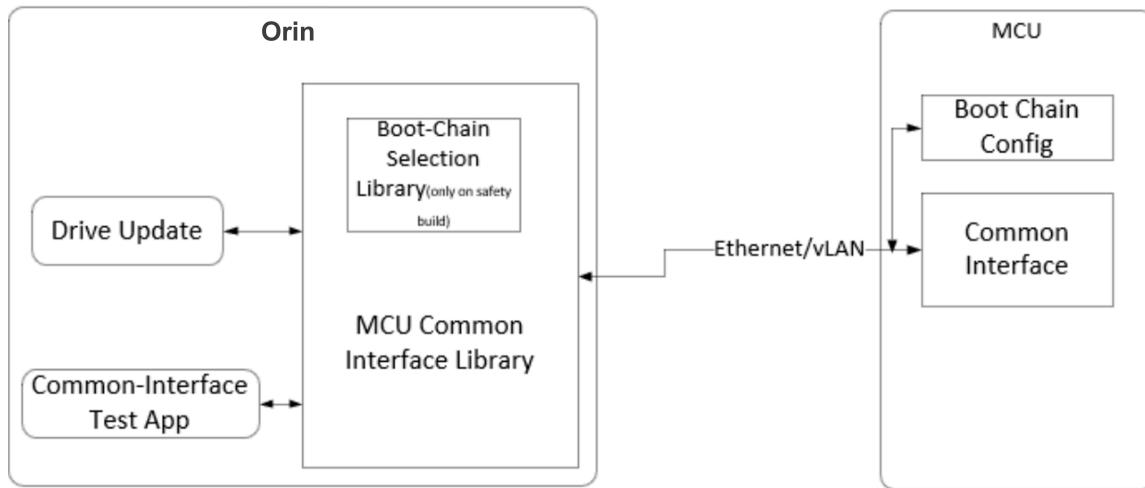
MCU manages power-on, power-off, recovery, and reset sequence of SoC on the NVIDIA DRIVE AGX™ board. It supports reflashing of its firmware and configuration, as well as GPIO-based boot chain selection of SoC firmware.

User applications on SoC, such as NVIDIA Drive® Update, communicate with MCU to use these features of MCU. This communication is established through Common Interface, which exposes a set of interfaces on SoC and connects to MCU using UDP over Ethernet.

Common Interface provides the following features:

- > Read InfoROM dump.
- > Get current version information of MCU firmware.
- > Validate MCU firmware to be flashed for compatibility.
- > Flash program Update and Production MCU firmware.
- > Set GPIO based boot chain configuration.
- > Read current configuration of GPIO-based boot chain configuration.
- > SoC reboot and MCU reboot.
- > Choose or get network configuration on MCU.
- > Test behavior of MCU boot chain APIs.
- > Test behavior of SoC reboot and MCU reboot.

The following diagram provides positioning of Common Interface modules in the system:



Library Usage and Configuration

User applications intending to use these functionalities shall link to Common Interface Library (libmcu_common_if.so).

Availability of these features depends on the build configuration. All features are available on the standard build. Boot chain APIs are available in safety builds only.

The Common Interface communication with MCU is only supported over a VLAN Ethernet interface; the VLAN interface is established at initialization.

Refer to the Configuration for Common Interface section for detailed information on the configuration parameters and VLAN interface creation.

Common Interface Sample Application Usage

The `common_if_testapp` is a command-line interface to verify supported features. Running the `common_if_testapp -h` command lists supported commands with brief description.

Examples of command with short description:

Resets MCU. Resets the complete DRIVE AGX board.

```
#common_if_testapp -mcureset
```

Reset Tegra x1.

```
#common_if_testapp -tegrareset x1
```

Set default boot chain of Tegra x1 to A.

```
#common_if_testapp -set_default_bootchain x1 A
```

Set the boot chain temporarily for next reboot.

```
#common_if_testapp -set_next_bootchain x1 B
```

Get the default boot chain for Tegra x1.

```
#common_if_testapp -get_default_bootchain x1
```

Get active/current boot chain for Tegra x1.

```
#common_if_testapp -get_active_bootchain x1
```

Flash given production firmware.

```
#common_if_testapp -flash_mcu_fw lib/firmware/DRIVE-V6.0.x-P3663-AFW-Aurix-StepB-5.xx.xx.hex
```

Flash given update firmware.

```
#common_if_testapp -flash_mcu_ufw lib/firmware/DRIVE-V6.x.x-P3663-NV-Aurix-UPDATE-StepB-1.xx.xx.hex
```

Read currently running firmware version.

```
#common_if_testapp -get_fw_version
```

Read update firmware version flashed on MCU.

```
#common_if_testapp -get_update_fw_version
```

Read the version of the hex file from its content.

```
#common_if_testapp -get_mcu_hexfile_version lib/firmware/DRIVE-V6.x.x-P3663-NV-Aurix-UPDATE-StepB-1.xx.xx.hex
```

Read data from infoROM dump.

```
#common_if_testapp -get_inforom_dump
```

Reboot MCU in Production FW mode.

```
#common_if_testapp -reboot_mcu_afw
```

Reboot MCU in Update-FW mode.

```
#common_if_testapp -reboot_mcu_ufw
```

Choose network Configuration on MCU.

```
#common_if_testapp -nw_cfg base
```

Get NW Configuration on MCU. Print the current status of MCU network configuration.

```
#common_if_testapp -nw_cfg getStatus
```

The additional commands listed with the -h(help) option are dummy and meant for debugging purposes only.

Executing SC7 Entry and Exiting from Console

- > On the NVIDIA DRIVE Orin™ console, run the `common_if_testapp -enter_sc7` command so that MCU starts the timer to wait for NVIDIA DRIVE Orin™ to enter SC7. This is done by monitoring the GPIO -SOC_PWR_REQ.
- > On the NVIDIA DRIVE Orin™ console, use the following API function to trigger SC7 entry:

```
status = nvdvms_set_vm_state(NVDVMS_SUSPEND) ;
if (status != NvDvmsSuccess) {
    /** Failure case **/
}
```

For more information, see the NvDVMS_Client::User Interface API module in the [API Reference](#).

- > As soon as NVIDIA DRIVE Orin™ enters SC7, pin SOC_PWR_REQ will be set to STD_HIGH, and MCU will execute SC7 entry sequence.
 - Full VRS sequence is not implemented. VM_EN1 will be pulled low.
- > If NVIDIA DRIVE Orin™ does not enter SC7 before the timeout of 20 seconds, MCU will go into error state. To recover from it, run either the `poweron` or `aurixreset` command.
- > To exit, run the command `exitsc7` on the MCU console.
 - AURIX_SOC_WAKE is toggled.

Running Safe-Shutdown from Console

For an orderly shutdown of NVIDIA DRIVE Orin™, when a shutdown request is triggered, MCU software waits for an indication of safe-shutdown completion from NVIDIA DRIVE Orin™ SoC through the GPIO pin (IST_DONE_N). If the indication is not received before a configured timeout (20 seconds), MCU software continues to power off and report errors to the user application.

To initiate safe shutdown:

- > On the NVIDIA DRIVE Orin™ console, run the `common_if_testapp -tegrareset` command to perform a tegrareset or the `common_if_testapp -mcureset` command to perform an Aurix reset.
- > On the NVIDIA DRIVE Orin™ console, use the following API function to trigger shutdown of TCF:

```
status = nvdvms_set_vm_state(NVDVMS_DEINIT) ;
if (status != NvDvmsSuccess) {
    /** Failure case **/
}
```

For more information, see the NvDVMS_Client::User Interface API module in the [API Reference](#).

- > After the request is successfully received by NvMCU_BootChainCfg on Aurix, acknowledgment will be sent. This request will be forwarded to the user application for further arbitration and if preconditions are met, NvMCU_OrinPwrCtrl will start the monitoring of Handshake GPIO.

- > As soon as the Handshake GPIO is set, NvMCU_OrinPwrCtrl will switch off the NVIDIA DRIVE Orin™ Module power rails and proceed with the hardware recommended shutdown sequence.
- > If the Handshake GPIO is not set within 20 seconds, NvMCU_OrinPwrCtrl will report an error to the user application and proceed with the same shutdown sequence.

3.8 Networking

3.8.1 Network Configuration



Note: The following section is not valid for P3898 as it has a different switch.

NVIDIA DRIVE® OS 6.0 supports two network configurations named Base and Safety. Base configuration is applicable to standard PCD configuration i.e., AV+L Standard and AV+Q Standard, while Safety is applicable for AV+Q prod, Prod_debug, and prod_debug_extra. These switch configurations are controlled by the AURIX MCU firmware via AURIX command line.

The configuration selection is enabled via AURIX MCU AUTOSAR firmware commands. Commands details:

```
getnetworkconfig : Get the current network config info
setnetworkconfig <config value>: Sets network config for next MCU boot
```

Where the <config value> parameter is as below:

Configuration	Value	Max MTU supported (MGBE2-OAK)	PTP support in OAK Switch	Switch Over the air update support
NW_CFG_BASE	0	Up to 16Kbytes	Yes (Refer PTP section)	Yes
NW_CFG_BASE_SAF	1	1500 bytes	Bypass (Switch CPU disable)	No

The major difference use cases between NW_CFG_BASE and NW_CFG_BASE_SAFETY are:

1. **Switch CPU & Firmware disabled** : NVIDIA switch development firmware only supports AVNU profile. For other profiles like AUTOSAR PTP support, this mode is helpful.
2. **Max MTU size**: The max MTU size of OAK switch is reduced to ~1500 to minimize the PTP bypass latency time when switch is not PTP aware.

3. **OTA support:** In switch CPU disable mode, switch firmware OTA is N/A as no firmware is running in the switch.



Note: It is important to keep the software in Tegra (Orin) and network configuration setting in Aurix compatible as defined above in table . This has to be done manually and there is no automated way to keep Orin software and Aurix 'setnetworkconfig' in sync/ compatible mode. Using a network configuration that does not match the Orin PCT might result in undefined behavior.

P3710 Networking Port Configuration

Networking port configuration is captured below and is common for both networking configurations i.e., NW_CFG_BASE and NW_CFG_BASE_SAFETY.

Node	Controller/ Switch Port	Mode & Speed	T1 Role	PHY present	L2 switch forwarding restriction
Tegra	EQOS	1G-BASE-T	NA	Yes	NA
	MGBE-0	10G-BASE-T1	Secondary	Yes	NA
	MGBE-1	10G-BASE-T	NA	Yes	NA
	MGBE-2	5G (XFI)	NA	No	NA
	MGBE-3	10G (XFI)	NA	No	NA
Switch-1 (88Q5072)	P0 (Internal CPU port)	1G	NA	No	No Restriction
	P1	100M-BASE- T1	Primary	Yes (internal)	No Restriction except forwarding to Switch-1 P8
	P2	100M-BASE- T1	Primary	Yes (internal)	No Restriction except forwarding to Switch-1 P8
	P3	100M-BASE- T1	Primary	Yes (internal)	No Restriction except forwarding to Switch-1 P8
	P4	100M-BASE- T1	Primary	Yes (internal)	No Restriction except forwarding to Switch-1 P8

Node	Controller/ Switch Port	Mode & Speed	T1 Role	PHY present	L2 switch forwarding restriction
	P5	100M-BASE-T1	Primary	Yes (internal)	No Restriction except forwarding to Switch-1 P8
	P6	100M-BASE-T1	Primary	Yes (internal)	No Restriction except forwarding to Switch-1 P8
	P7	100M-BASE-T1	Primary	Yes (internal)	No Restriction except forwarding to Switch-1 P8
	P8	1G (RGMII)	NA	No	Only to P10
	P9	1G-BASE-T1	Primary	Yes	No Restriction except forwarding to Switch-1 P8
	P10	5G (XFI)	NA	No	No Restriction
	P11	PCIe Gen3 X1	NA	No	No Restriction except forwarding to Switch-1 P8
Switch-2 (88Q6113)	P0 (Internal CPU port)	1G	NA	No	No Restriction
	P1	1G-BASE-T1	Secondary	Yes	No Restriction
	P2	1G-BASE-T1	Secondary	Yes	No Restriction
	P3	1G-BASE-T1	Secondary	Yes	No Restriction
	P4	1G-BASE-T1	Secondary	Yes	No Restriction
	P5	1G-BASE-T1	Secondary	Yes	No Restriction
	P6	SGMII	NA	No	No Restriction
	P7	1G-BASE-T1	Secondary	Yes	No Restriction
	P8	1G-BASE-T1	Secondary	Yes	No Restriction

Node	Controller/ Switch Port	Mode & Speed	T1 Role	PHY present	L2 switch forwarding restriction
	P9	1G-BASE-T1	Secondary	Yes	No Restriction
	P10	10G (XFI)	NA	No	No Restriction
	P11	PCIe Gen3 X2	NA	No	No Restriction
Tegra PCIe Ethernet	LAN 7431 controller	1G-BASE-T1	Master	Yes	NA

P3663 Network Port Configuration

Networking port configuration is captured below and its common for both networking configurations i.e., NW_CFG_BASE and NW_CFG_BASE_SAFETY.

Node	Controller/ Switch Port	Mode & Speed	T1 Role	PHY present	L2 Routing
Tegra	EQOS	1G-BASE-T1	NA	Yes	NA
	MGBE-0	10G-BASE-T1	Secondary	Yes	NA
	MGBE-2	5G (XFI)	NA	No	NA
Switch-1 (88Q5072)	P0 (Internal CPU port)	1G	NA	No	No Restriction
	P1	100M-BASE- T1	Primary	Yes (internal)	No Restriction except forwarding to Switch-1 P8 (ie, to Aurix)
	P2				
	P3				
	P4				
	P5				
	P6				
	P7				
	P8	1G (RGMII)	NA	No	Only to P10 (ie, Aurix to Tegra)

Node	Controller/ Switch Port	Mode & Speed	T1 Role	PHY present	L2 Routing
	P10	5G (XFI)			No Restriction
	P11	PCIe Gen3 X1			No Restriction except forwarding to Switch-1 P8 (ie, to Aurix)
Tegra PCIe Ethernet	LAN 7431 controller	1G-BASE-T1	Primary	Yes	NA

P3898 Network Port Configuration

The p3898 network port configuration is as follows:

Node	Controller/ Switch Port	Mode & Speed	T1 Role	PHY present	L2 Routing
Tegra	EQOS	1G (RGMII)	N/A	No	N/A
Switch	P2	100M-BASE-T1	Primary	Yes (internal)	No Restriction except forwarding to Switch P5 (I.e., to AURIX)
	P3				
	P4				
	P5	1G (RGMII)	N/A	No	Only to P8 (I.e., Aurix to Tegra)
	P8				No restriction
	P0	1G-BASE-T1	Primary	Yes	No Restriction except forwarding to Switch P5 (I.e., to AURIX)
	P6				

3.8.2 Static IP Assignment

Ubuntu 17.10, 18.04, and higher follow the steps below to assign the static IP to DUT's 1G or 10G interface.

1. Create .yaml config file for netplan in /etc/netplan if it does not exist. A number prefix is used for orderly processing. For more information, refer to `man netplan`.



Note: Replace eth0 with your system's interface name

```
sudo cat >/etc/netplan/01-network-manager-all.yaml
network:
    version: 2
    renderer: networkd
    ethernets:
        eth0:                                     # Use your interface
            name
            dhcp4: no
            dhcp6: no
            addresses:
                - 10.0.0.X/22      # Your IP address
            gateway4: 10.0.0.1          # Your gateway
            nameservers:
                search: [nvidia.com, client.nvidia.com,
                          nvclient.nvidia.com] #Your corp domain names
                addresses: [DNS1,DNS2]
```

2. Once the editing of `01-network-manager-all.yaml` is completed, apply the settings:

```
sudo netplan apply
```

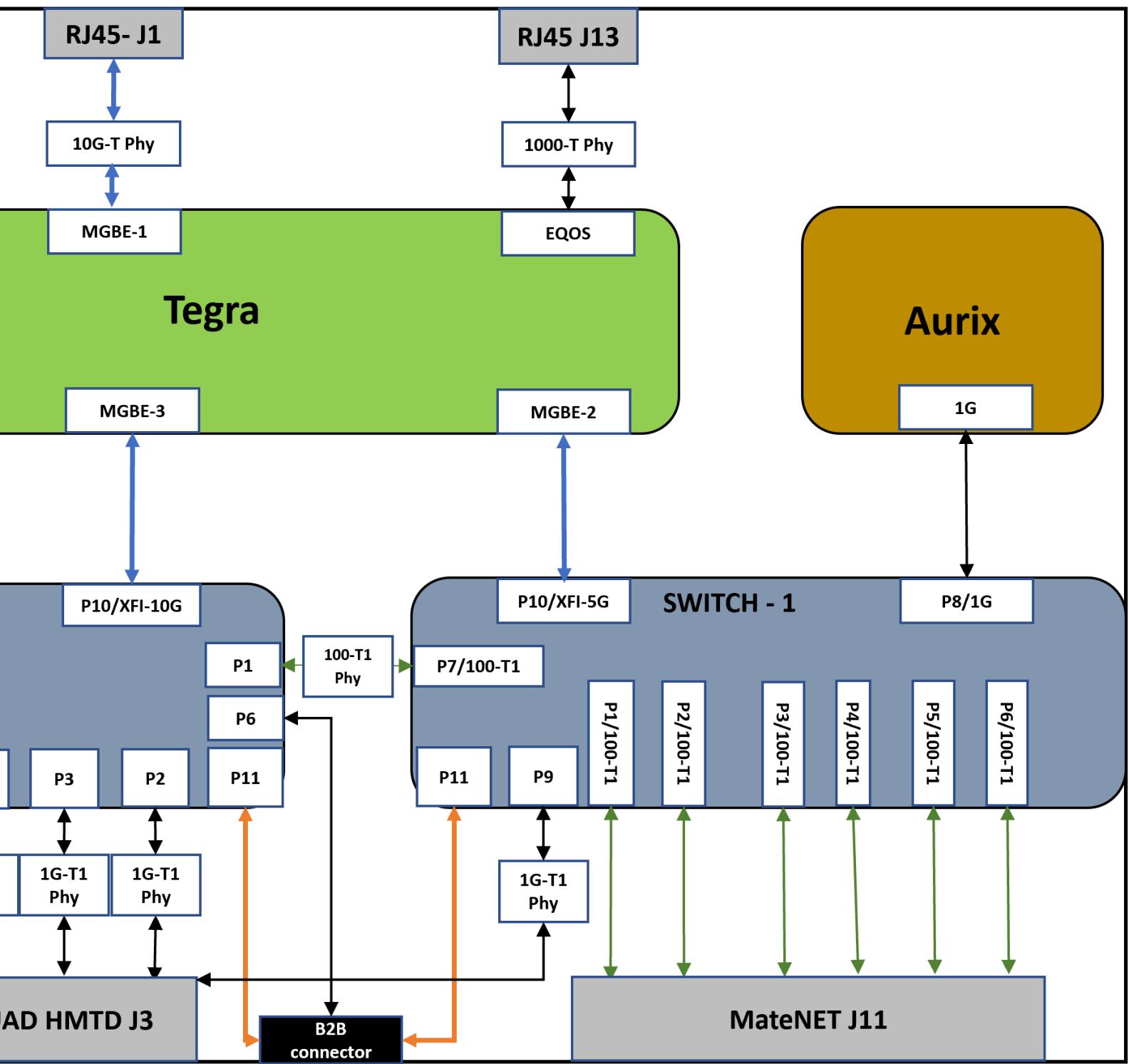
3. To review the configuration:

```
ip addr
ip show
```

Clearly mention the windowing system used (X, Wayland ,lightdm, etc.).

3.8.3 P3710 Networking

The following diagram shows the P3710 network topology:

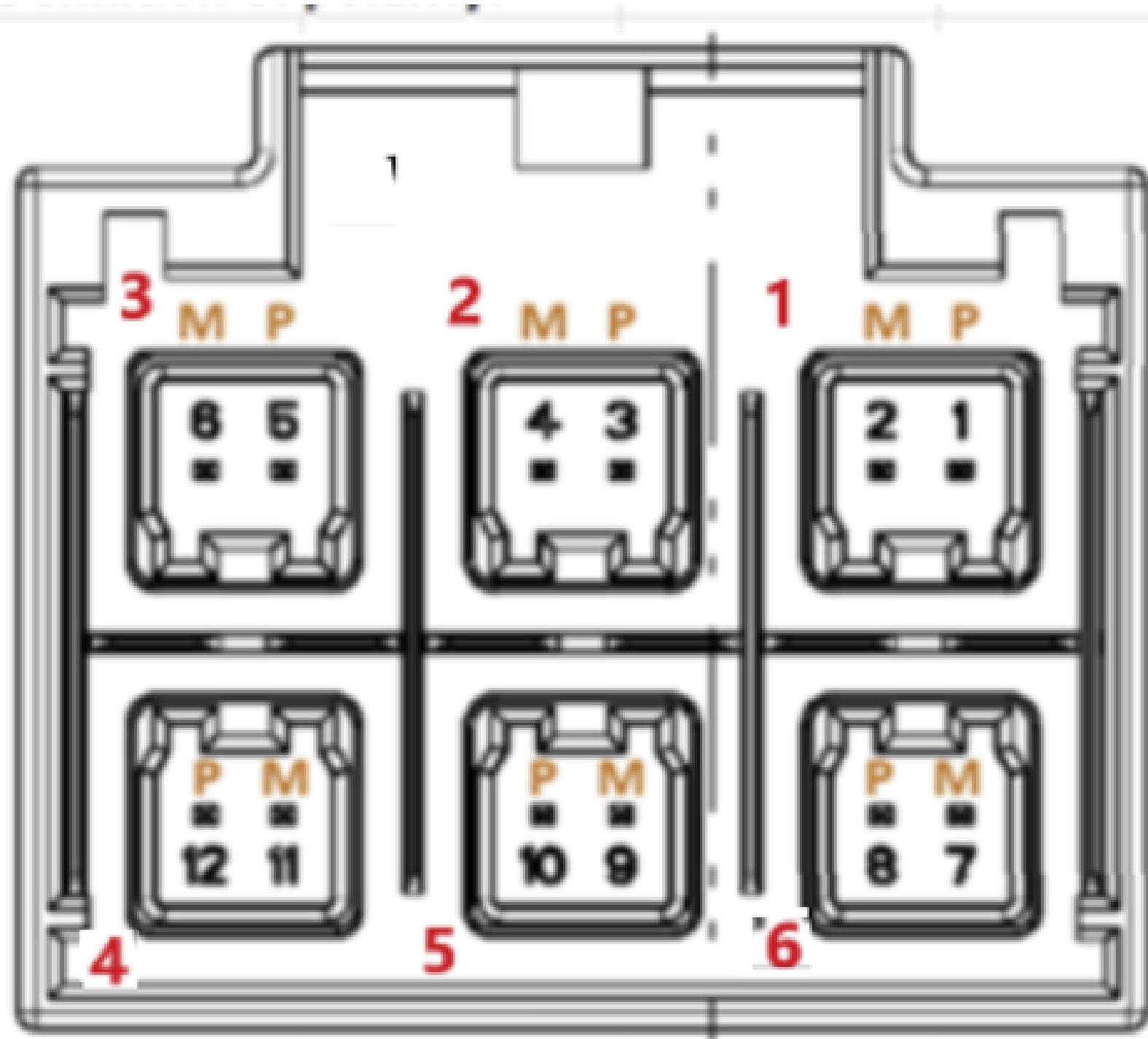


Board External Connectors

(Common for both P3663 and P3710)

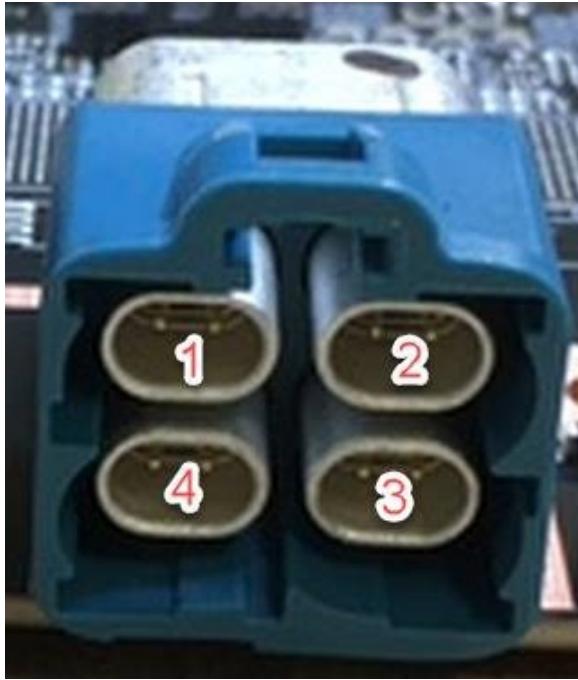
Matenet

Matenet connector has connection for Port 1 to Port 6 of Oak switch.



QUAD HMTD

The following is the channel map of HMTD:



External Connector Port: Mapping to Switch/Controller

QUAD HMTD Connector J3

Port	Connected to
1	Switch-1 (88Q5072) Port 9
2	Switch-2 (88Q6113) Port 2
3	Switch-2 (88Q6113) Port 3
4	Switch-2 (88Q6113) Port 4

QUAD HMTD Connector J4

Port	Connected to
1	Switch-2 (88Q6113) Port 5
2	Switch-2 (88Q6113) Port 7
3	Switch-2 (88Q6113) Port 8
4	Switch-2 (88Q6113) Port 9

Dual HMTD Connector J2

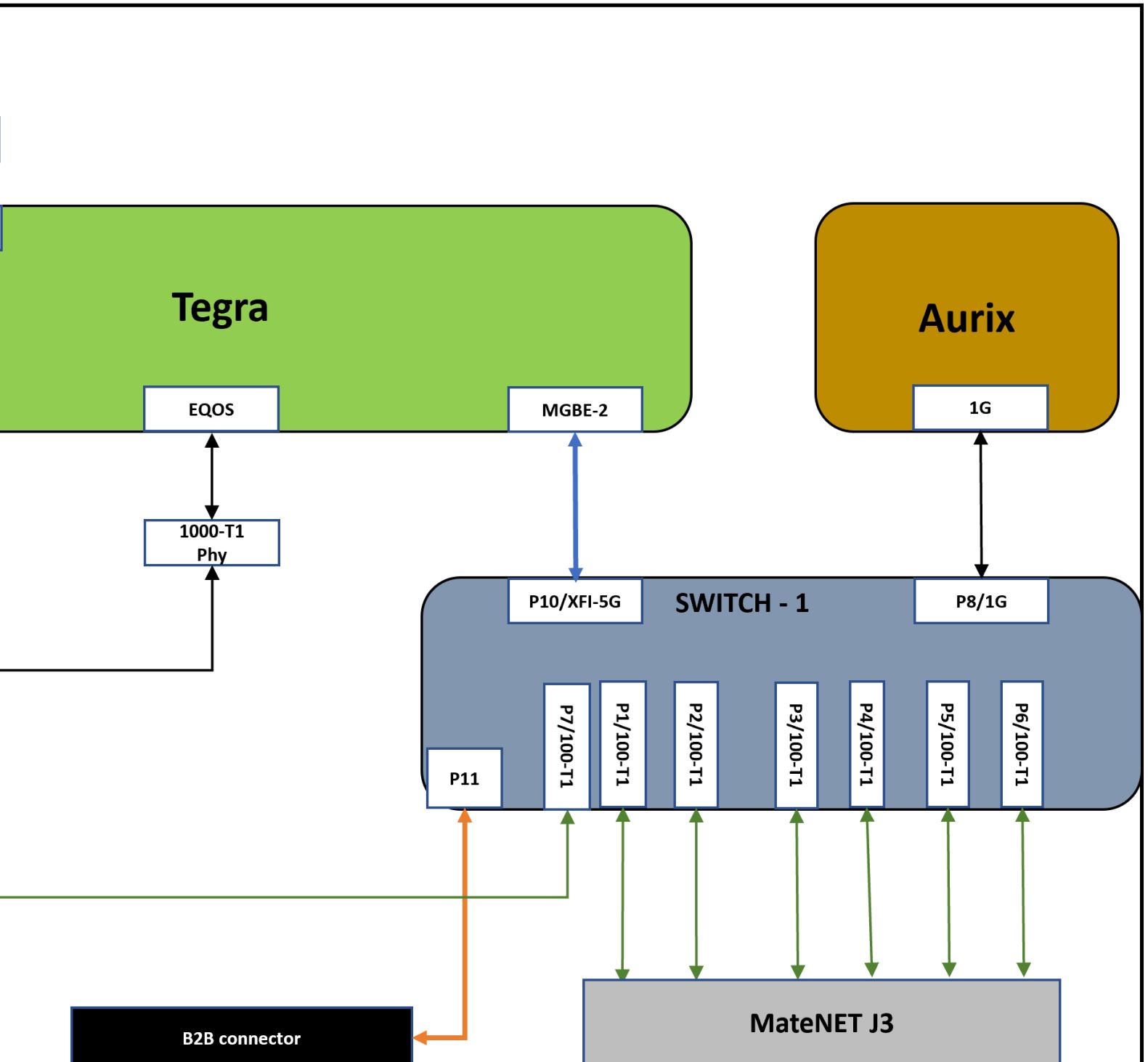
Port	Connected to
1	Tegra MGBEO Ethernet Controller
2	LAN7431 (PCIe Ethernet Controller)

MateNET Connector J11:

Port	Pin Number	Connected to
1	Pin 1&2	Switch (88Q5072) Port 1
2	Pin 3&4	Switch (88Q5072) Port 2
3	Pin 5&6	Switch (88Q5072) Port 3
4	Pin 11&12	Switch (88Q5072) Port 4
5	Pin 9&10	Switch (88Q5072) Port 5
6	Pin 7&8	Switch (88Q5072) Port 6

3.8.4 P3663 Networking

The following diagram shows the P3663 network topology:



External Connector Channels - Mapping to Switch/Controller

QUAD HMTD Connector J7

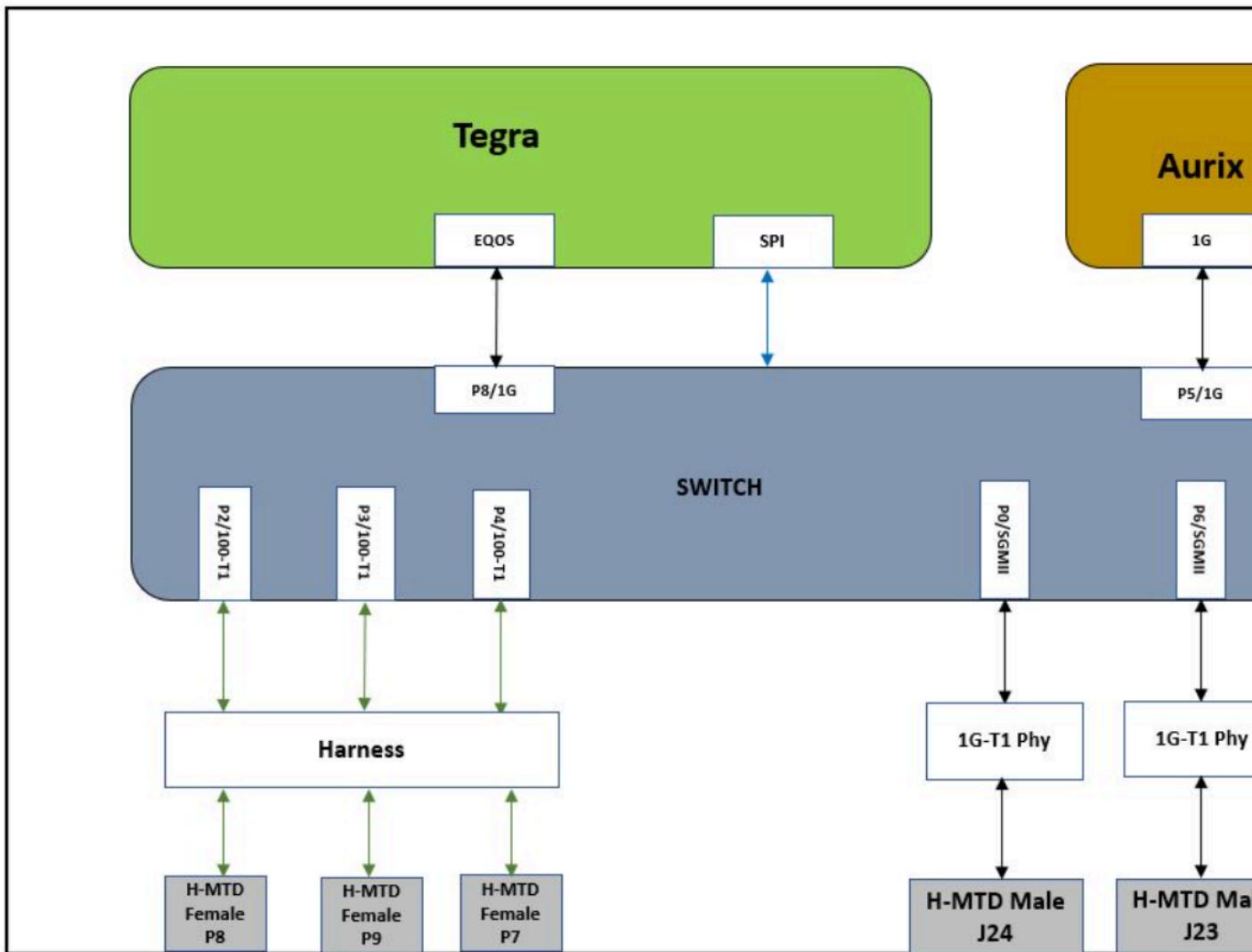
Port	Connected to
1	Switch-1(88Q5072) Port 7
2	LAN7431 (PCIe Ethernet Controller)
3	Tegra MGBEO Ethernet Controller
4	Tegra EQOS Ethernet Controller

MateNET Connector J3

Port	Pin Number	Connected to
1	Pin 1&2	Switch (88Q5072) Port 1
2	Pin 3&4	Switch (88Q5072) Port 2
3	Pin 5&6	Switch (88Q5072) Port 3
4	Pin 11&12	Switch (88Q5072) Port 4
5	Pin 9&10	Switch (88Q5072) Port 5
6	Pin 7&8	Switch (88Q5072) Port 6

3.8.5 3898 Networking

The following diagram shows the P3898 network topology:



External Connector Channels : Mapping to Switch/Controller

Switch Port	Connected To
P8	Tegra EQOS Ethernet Controller
P5	Aurix Ethernet Controller
P2	Out of Harness, H-MTD Female cable tagged P8
P3	Out of Harness, H-MTD Female cable tagged P9
P4	Out of Harness, H-MTD Female cable tagged P7
P0	H-MTD Male connector J24
P6	H-MTD Male connector J23

Change to Enable Networking with AFW

By default, the p3898 switch is configured to work with AURIX IFW. In order for Tegra and AURIX communication over VLAN to work with Aurix AFW, nv_tacp_init.sh script must be updated.

- > The nv_3898_vlan_setup.sh script (called from nv_tacp_init.sh) that configures the p3898 switch for VLAN.
- > When one argument is passed, the script configures the switch to work with AURIX IFW. If no argument is passed, the switch is configured to work with Aurix AFW.
- > Remove the argument passed to the nv_3898_vlan_setup.sh script (i.e., remove ifw) to make it work with AURIX AFW.

3.8.6 VLAN Configuration

IBF VLAN aka VLAN 200 AKA TACP VLAN

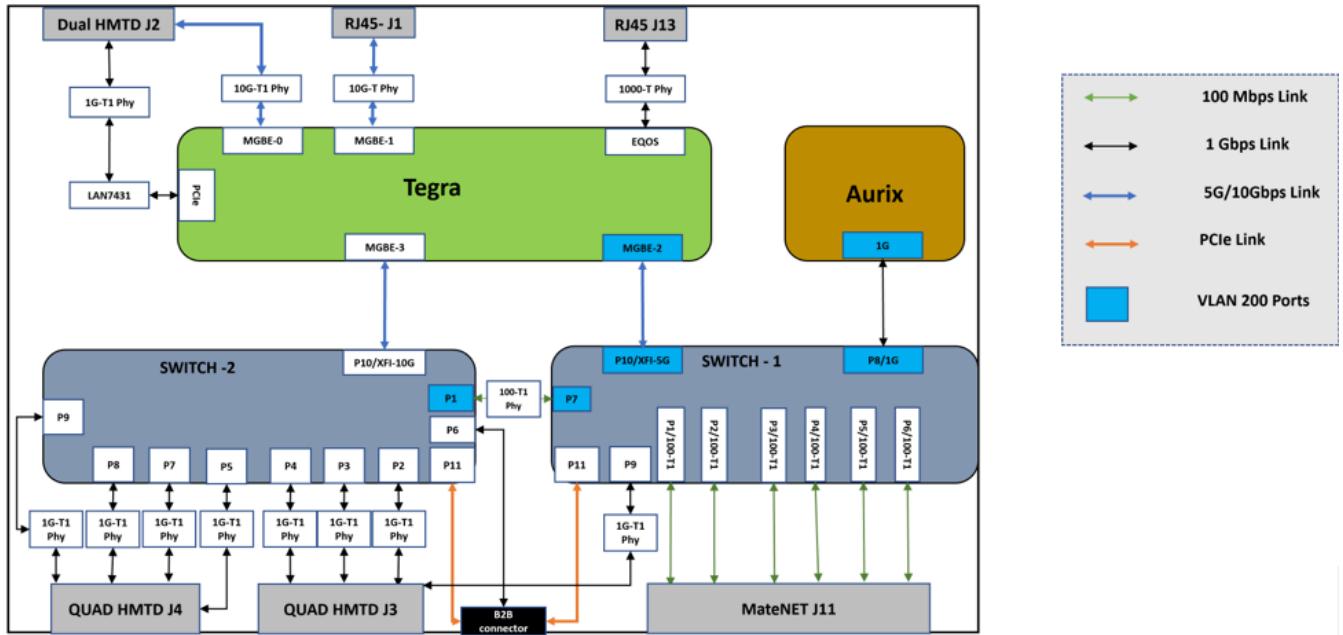
Intra-board function VLAN is used for communication between onboard devices like Aurix, Tegra, and switches. No Ethernet frames over this VLAN go outside the board. IBF VLAN is used for the following:

1. Tegra-Tegra communication
2. Tegra-Aurix communication
3. Aurix OTA from Tegra
4. Switch Firmware over-the-air (OTA) updates

VLAN 200 Config in P3710

In 3710, the following ports and controllers are part of VLAN 200.

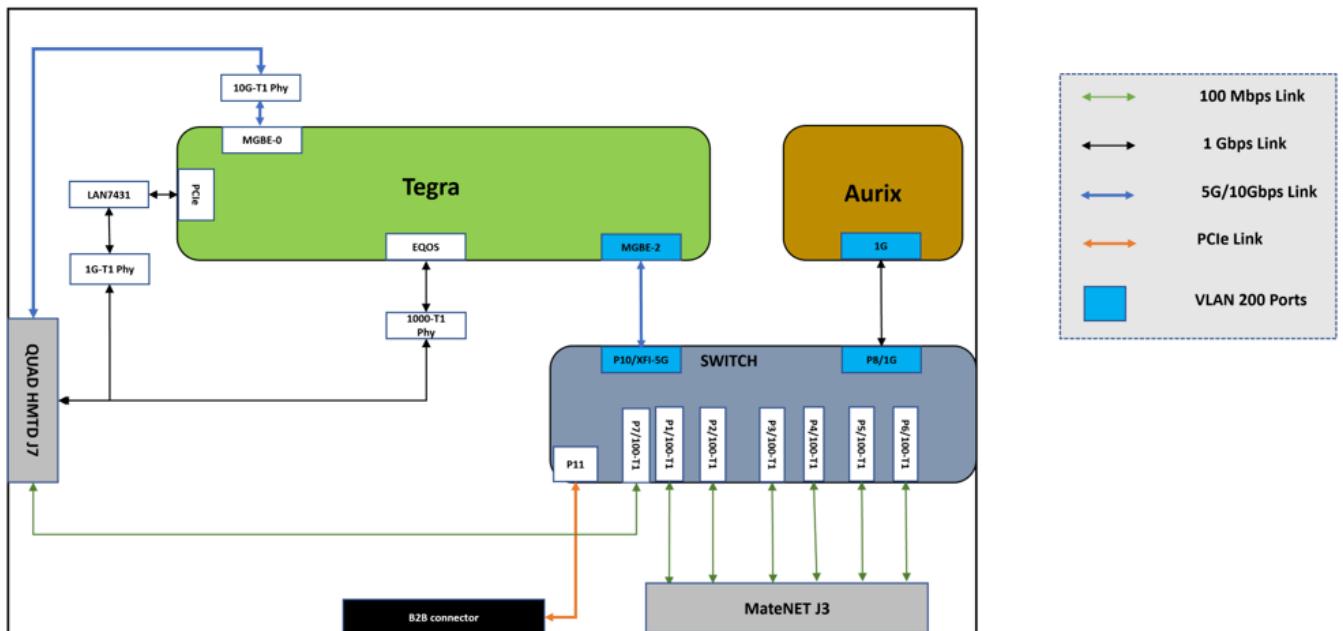
- > Tegra MGBE-2 interface
- > Aurix Ethernet interface
- > Switch-1: P0 (Internal CPU) , P10 (interfacing MGBE-2), P8 (Aurix interfacing port) and P7 (Switch-2 interfacing port)
- > Switch-2: P0 (Internal CPU) and P1 (Switch-2 interfacing port)



VLAN 200 Config in P3663

In 3663, the following ports and controllers are part of VLAN 200.

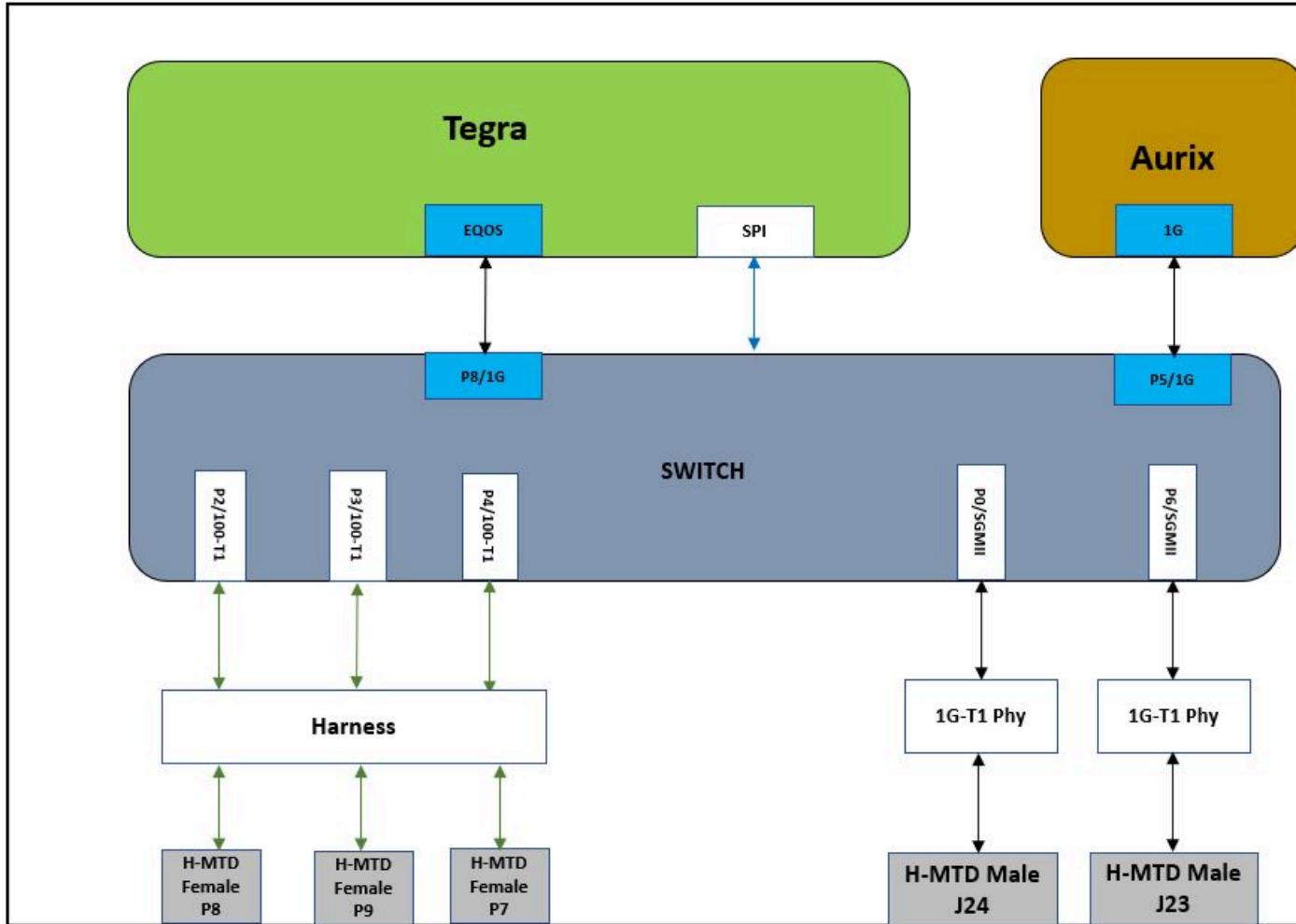
- > Tegra MGBE-2 interface
- > Aurix Ethernet interface
- > Switch-1: P0 (Internal CPU), P10 (interfacing MGBE-2) and P8 (Aurix interfacing port)



VLAN 200 Config in P3898

In 3898, the following ports and controllers are part of VLAN 200.

- > Tegra EQOS interface
- > Aurix Ethernet interface
- > Switch: P8 (interfacing EQOS) and P5 (AURIX interfacing port)



TACP Networking VLAN Configuration

The VLAN networking configuration to set up this channel occurs in three distinct places in standard or safety operating systems.

1. Tegra (Orin) side configuration
2. Network switch configuration
3. Aurix AUTOSAR configuration

Tegra (Orin) Side Configuration

For standard OS the TACP VLAN configuration parameters are stored in the device tree while the safety OS has its values within a configuration file.

Standard OS

For standard OS the TACP VLAN configuration parameters are stored in the device tree, which has files located here:

```
<top>/hardware/nvidia/platform/t23x/automotive/kernel-dts/
```

They are further broken down by the board type. The networking information is in the following device tree files under the node `board_config`.

```
p3710/common/tegra234-p3710-0010.dtsi  
p3663/common/tegra234-p3663-0001.dtsi  
p3898/common/tegra234-p3898-0010.dtsi
```

The configuration file supports these parameters under the board config node required for the TACP VLAN setup.

- > Tegra (Orin) IP address
- > VLAN ID and interface name

```
board_config {
    TEGRA_IP_ADDRESS="10.42.0.28";
    aurix {
        /* Provide instance.200 as aurix interface,
         * where instance is network controller
         * interface name */
        aurix_linux_interface="mgbe2_0.200";
        aurix_qnx_interface="mgbe2_0.200";
        AURIX_IP_ADDRESS="10.42.0.146";
    };
}
```

Where `aurix_qnx_interface/ aurix_linux_interface = <MAIN_ETH_IFACE>.<VLANID>` : `<MAIN_ETH_IFACE>` is the network interface over the VLAN is supposed to created and `<VLANID>` is the vlan ID of the newly created VLAN interface. `TEGRA_IP_ADDRESS= <IP_ADDRESS>` : IP address for Tegra tacp VLAN interface

Tegra-side TACP VLAN Customization

Tegra side TACP VLAN customization is possible for the following parameters by changing the value assigned in the board-specific device tree in standard OS, and by changing the hardcoded values in the bootup script.

- > Tegra IP address
- > VLAN ID and interface name



Note: Changes in VLAN parameters requires corresponding changes in the Aurix AFW for new VLAN support along with the new VLAN configuration in the network switches. Changes in IP address (if other subnet) requires a change in IP address of the VLAN interface on the Aurix MCU side.

Network Switch VLAN Configuration



Note: For the P3898 board, switch configuration occurs via Orin over SPI or through the network switch firmware.

In NVIDIA platforms, network switch configuration occurs via Aurix MCU firmware or through the network switch firmware. These configuration cannot be altered in run time.

Aurix MCU VLAN Configuration

Refer to the *MCU Porting Guide* for details.

3.8.7 Time Synchronization

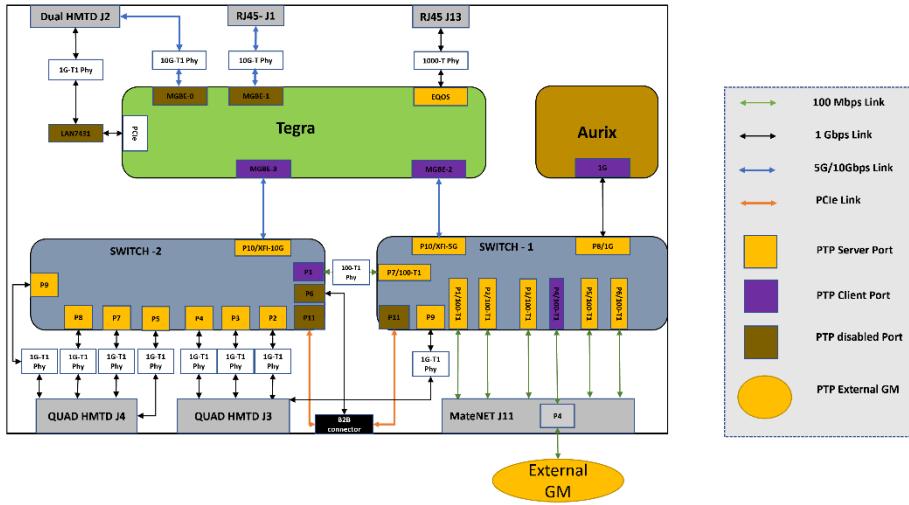
Platform time sync essentially represents the PTP topology and PTP profile supported for development and production use cases.

Supported Profiles

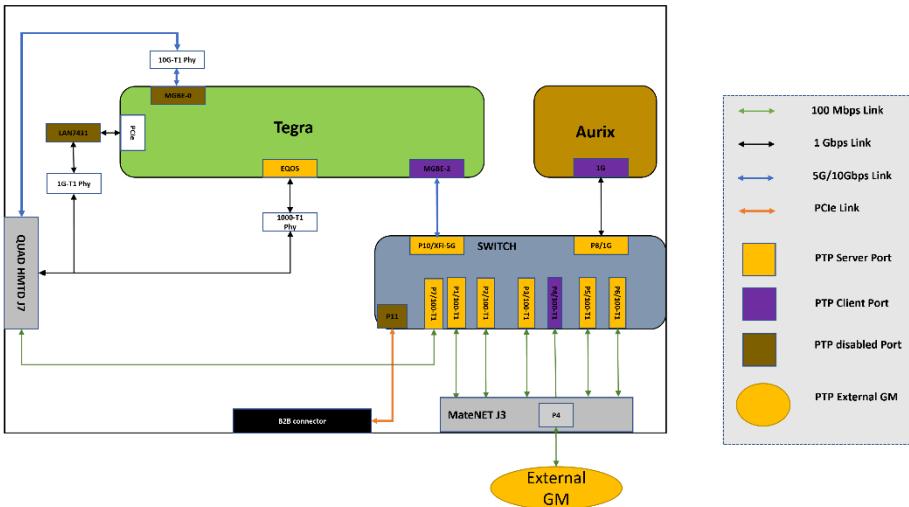
Supported Profile	Comments
AVNU CDS 1.6	<p>With time aware switch</p> <p>All networking nodes of the DRIVE development platforms are made time-aware systems/nodes, such as the Tegra CPU nodes, which act as time-aware end-stations, and all switches acts as time-aware bridge (Only AVNU CDS 1.6 profile).</p> <p>Orin development platforms have total 2 different models of Ethernet switches connected on board.</p> <ul style="list-style-type: none"> > Marvell 88Q5072 switch (OAK). > Marvell 88Q6113 switch (Spruce). <p>Both the switches has its own internal MCU and firmware capable of handling the PTP (AVNU CDS 1.6 profile) locally in side switch. Hence switches are made time-aware bridge by enabling the PTP support in its firmware. All the external ports coming out of switches supports PTP with static PTP roles.</p>
AUTOSAR Time Synchronization Protocol (pdelay disabled)	<p>With Switch PTP Bypass (Switch Internal CPU disabled). This can be achieved by using networkconfig 1 from AURIX MCU AUTOSAR firmware.</p>

Platform PTP Topology and Port Roles

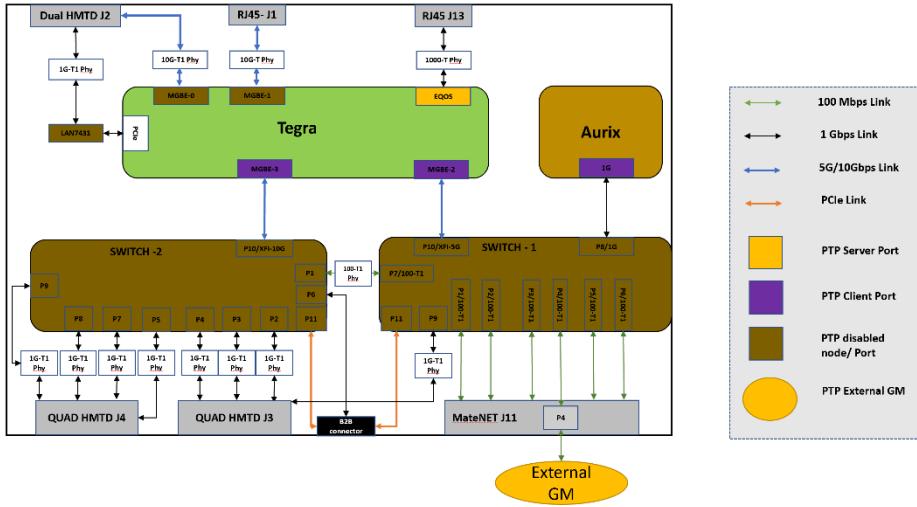
3710 : NW_CFG_BASE



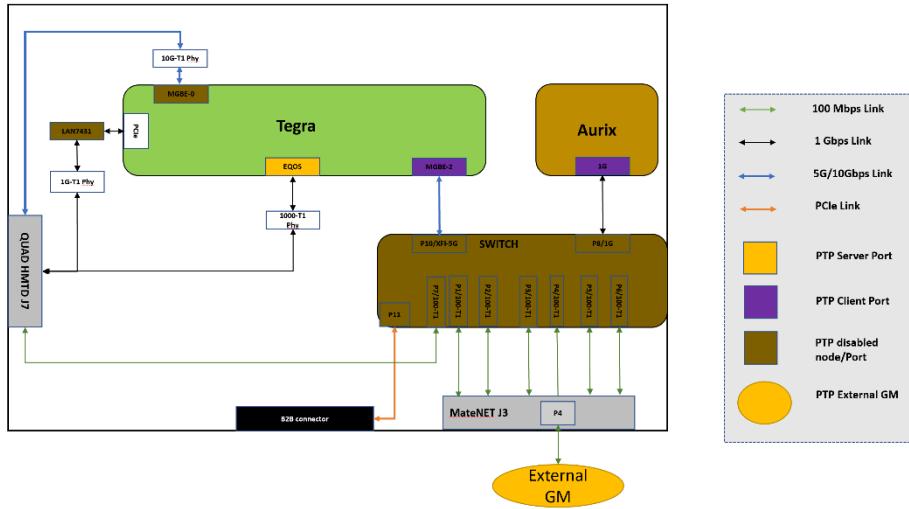
3663: NW_CFG_BASE



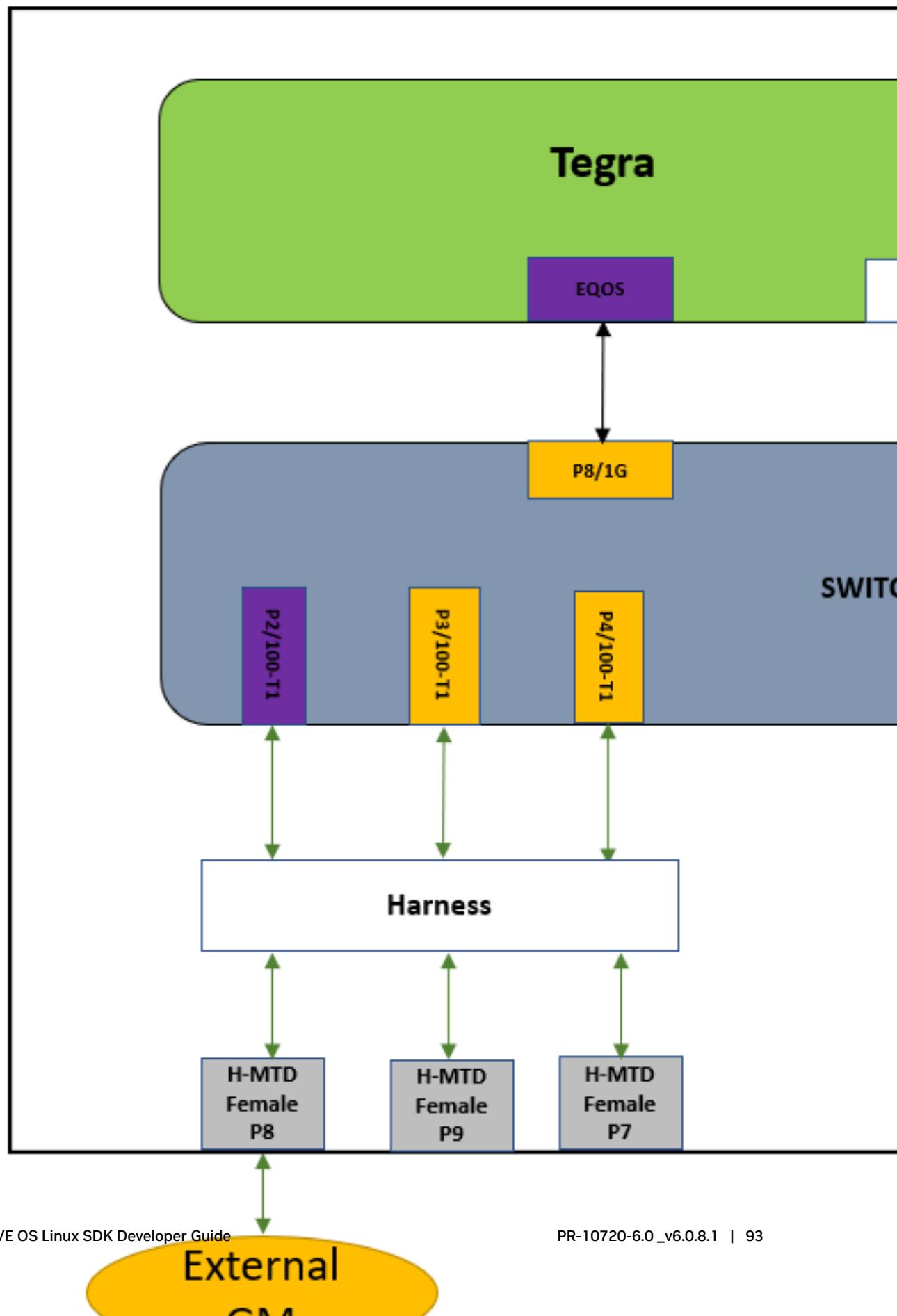
3710: NW_CFG_BASE_SAFETY



3663 : NW_CFG_BASE_SAFETY



3898: Platform PTP Port Roles



Port Roles

3710 : Platform PTP Port Roles

Platform Ports (Internal/ External)	Connecting External Port/ Internal Node	Development Use Case			Production Use Case		
		Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/ protocol)	Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/ protocol)
Switch-1 (88Q5072) - Port 1	MateNET Connector J11: P1	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-1 (88Q5072) - Port 2	MateNET Connector J11: P2	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-1 (88Q5072) - Port 3	MateNET Connector J11: P3	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-1 (88Q5072) - Port 4	MateNET Connector J11: P4	Secondary	AVNU AutoCDS v1.6	Primary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-1 (88Q5072) - Port 5	MateNET Connector J11: P5	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-1 (88Q5072) - Port 6	MateNET Connector J11: P6	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)

Platform Ports (Internal/External)	Connecting External Port/ Internal Node	Development Use Case			Production Use Case		
		Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/ protocol)	Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/ protocol)
Switch-1 (88Q5072) - Port 9	QUAD HMTD Connector J3: P1	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-2 (88Q6113) - Port 2	QUAD HMTD Connector J3: P2	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-2 (88Q6113) - Port 3	QUAD HMTD Connector J3: P3	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-2 (88Q6113) - Port 4	QUAD HMTD Connector J3: P4	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-2 (88Q6113) - Port 5	QUAD HMTD Connector J4: P1	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-2 (88Q6113) - Port 7	QUAD HMTD Connector J4: P2	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)

Platform Ports (Internal/External)	Connecting External Port/ Internal Node	Development Use Case			Production Use Case		
		Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/ protocol)	Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/ protocol)
Switch-2 (88Q6113) - Port 8	QUAD HMTD Connector J4: P3	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-2 (88Q6113) - Port 9	QUAD HMTD Connector J4: P4	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-1 (88Q5072) - Port 10	Tegra-MGBE 2	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-2 (88Q6113) - Port 10	Tegra-MGBE 3	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch-1 (88Q5072) - Port 8	Aurix - MAC	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Tegra EQOS	RJ45 -J13	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Primary	AUTOSAR TSync (pdelay disabled)	Secondary/ AUTOSAR TSync (pdelay disabled)

3663 : Platform PTP Port Roles

Table 1.

Platform Ports (Internal/External)		Development Use Case			Production Use Case		
		Connecting External Port / Internal Node	Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/protocol)	Port PTP Role	Protocol Support
Switch (88Q5072) - Port 1	MateNET Connector J3: P1	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch (88Q5072) - Port 2	MateNET Connector J3: P2	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch (88Q5072) - Port 3	MateNET Connector J3: P3	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch (88Q5072) - Port 4	MateNET Connector J3: P4	Secondary	AVNU AutoCDS v1.6	Primary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch(88Q5072) - Port 5	MateNET Connector J3: P5	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch (88Q5072) - Port 6	MateNET Connector J3: P6	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)

Platform Ports (Internal/External)	Connecting External Port / Internal Node	Development Use Case			Production Use Case		
		Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/protocol)	Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/protocol)
Switch (88Q5072) - Port 7	QUAD HMTD Connector J7: P1	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch (88Q5072) - Port 10	Tegra-MGBE 2	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Switch (88Q5072) - Port 8	Aurix - MAC	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Disabled	NA	Secondary/ AUTOSAR TSync (pdelay disabled)
Tegra EQOS	QUAD HMTD Connector J7: P4	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Primary	AUTOSAR TSync (pdelay disabled)	Secondary/ AUTOSAR TSync (pdelay disabled)

3898 : Platform PTP Port Roles

Table 2.

	Connecting External Port / Internal Node	Development Use Case			Production Use Case	
		Port PTP Role	Protocol Support	PTP device allowed to connect to this port (role/protocol)	Port PTP Role	Protocol Support
3898 Switch - Port 2	Harness H-MTD P8	Secondary	AVNU AutoCDS v1.6	Primary/ AVNU AutoCDS v1.6	Secondary	N/A
3898 Switch - Port 3	Harness H-MTD P9	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Primary	N/A
3898 Switch - Port 4	Harness H-MTD P7	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Primary	N/A
3898 Switch - Port 0	Board H-MTD J24	Primary	AVNU AutoCDS v1.6	Primary/ AVNU AutoCDS v1.6	Primary	N/A
3898 Switch - Port 6	Board H-MTD J23	Primary	AVNU AutoCDS v1.6	Secondary/ AVNU AutoCDS v1.6	Primary	N/A

**Note:**

The AURIX MCU PTP role shown in above diagram is for representation purpose only (to make the topology complete). DRIVE OS does not have PTP enabled in AURIX firmware.

3.8.7.1 Orin Time Sync

Orin Time Sync has multiple components as described below in diagram. It consist of:

PTP Support in Orin

As per the supported PTP topology, One of the Tegra MAC and its associated PHC needs to get synchronized with external Grand Master clock. Once synchronized this MAC PHC (PTP hardware counter) works as a base for:

- > All the Tegra internal clocks which needs to be synchronized or to be correlated as per the use-case.
- > All the software component requiring the current PTP time in system.

AVNU PTP Support

In DRIVE OS Linux (AV+L), Linuxptp daemon i.e ptp4l is supposed to be run in client mode with automotive profile mode to synchronize Tegra's MGBE_2 mac interface from external GM.

```
./ptp4l -f automotive_slave.cfg -i mgbe2_0 -p /dev/ptp<X> -m
// where:
// /dev/ptp<X> - PTP dev node associated with the mgbe2_0
```

For device connected on EQOS another daemon instance can be started with following command in server mode. To identify <x>, use the ethtool -T [Interface ID] command. If ethtool is not installed on the host and DUT, use the sudo apt install ethtool command to install.

```
./ptp4l -f automotive_master.cfg -i eqos_0 -p /dev/ptp<X> -m
// where:
// /dev/ptp<X> - PTP dev node associated with the eqos_0
```

AUTOSAR TSync PTP Support



Note: NVIDIA provided Tsync PTP implementation (i.e., gptpTsync) is for development purpose only and it must not be used for production. The 3898 platform does not support AUTOSAR TSync PTP; only AVNU CDS PTP is supported.

AUTOSAR Ethernet Tsync profile PTP commands:

```
/ #Server Mode
cd /opt/nvidia/drive-linux/samples/nvavb/daemons/
sudo ./gptpTsync <interface> -F autosar_cfg.ini -INITPDELAY 0 -0PERPDELAY 0 -INITSYNC -3
-OPERSYNC -3 -S -V -GM

// #client Mode
cd /opt/nvidia/drive-linux/samples/nvavb/daemons/
sudo ./gptpTsync <interface> -F autosar_cfg.ini -INITPDELAY 0 -0PERPDELAY 0 -INITSYNC -3
-OPERSYNC -3 -S -V -N -A

// where:
// <iface> : Interface name i.e eqos_0, mgbe0_0, mgbe1_0 on which PTP is supposed to be
started. It could be a VLAN interface above the primary interface.
```

The following section explains the autosar specific configurable parameters from INI file for gptpTsync binary:

```
/*
# Time secure TLV related data
# Specification for 10.2.4 EthTSynGlobalTimeFollowUpDataIDList, Also refer
ECUC_EthTSyn_00030
# Provide 16 data element values in decimal
# _Provide value in decimal value only(not in hexadecimal)_ */
FollowUpDataIdList = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

/*
# Time secure TLV CRC_Time_Flags for CRC calcuation in master mode.
# Refer CRC_Time_Flags from SWS_EthTSyn_00065
# Following values to be 'OR'ed to create flag as per requirement
# BitMask 0x01 [messageLength]
# BitMask 0x02 [domainNumber]
# BitMask 0x04 [correctionField]
# BitMask 0x08 [sourcePortIdentity]
# BitMask 0x10 [sequenceId]
# BitMask 0x20 [preciseOriginTimestamp]
# Value provided in below config will be 'AND' with 0x3F to consider only valid bits
# _Provide value in hex format only_ */
EthTSynCrcTimeFlagsTxSecured = 0x37

/*
# Status TLV related data
# if SyncToGateWay is set, SGW with SyncToSubDomain will be set. Otherwise, SyncToGTM
will be set.
# Valid values 0 and 1
# _Provide value in decimal value only(not in hexadecimal)_ */
SyncToGateWay = 0

/*
# User data TLV number of bytes UserDataLength
# this value should be 1 >= UserDataLength <= 3 */
UserDataLength = 1

/*
# User data TLV Related data
# Provide user byte of size uint8_t with space separated, i.e UserByte = UserByte_0
UserByte_1 UserByte_2
# _Provide value in decimal value only(not in hexadecimal)_ */
UserByte = 0 0 0

/*
# CRC validation mode for slave
# CRC_IGNORED and CRC_VALIDATED */
EthTSynRxCrcValidated = CRC_VALIDATED

/*
# Static Pdelay setting.
# Set to 1 to disable path delay calculation using Pdelay_req, Pdelay_resp,
Pdelay_respup.
```

```

# and use EthTSynGlobalTimePropagationDelay as static Link delay.
# Setting it to 1 will also make daemon to ignore path delay messages received.

# Set 0 to enable path delay calculation using Pdelay_req, Pdelay_resp, Pdelay_respfup
# and ignore EthTSynGlobalTimePropagationDelay.
# _Provide value in decimal value only(not in hexadecimal)_ */
UseStaticPdelay = 0

/*
# Static Delay in nanosecond will be used when UseStaticPdelay is set to 1
# Default value of 0 will be used if UseStaticPdelay set and below field is not
# populated.
# _Provide value in decimal value only(not in hexadecimal)_ */
EthTSynGlobalTimePropagationDelay = 20000000

/*
# Option to enable and disable processing and transmit of sub-TLV's.
# Set values to 1 enable and 0 to disable
# Time secure sub-TLV
# _Provide value in decimal value only(not in hexadecimal)_ */
EthTSynTLVFollowUpTimeSubTLV = 1
/* # Status sub-TLV */
EthTSynTLVFollowUpStatusSubTLV = 1
/* # User data sub-TLV */
EthTSynTLVFollowUpUserDataSubTLV = 1

```

Other PTP Profiles in Orin

DRIVE OS Linux supports IEEE 802.1AS profile. Support for this profile is limited to the software stack running in Tegra/Orin i.e., no support in switches. This means this profile can not be enabled on Ethernet MACs, which are directly connected to the switches on the DRIVE Orin Development Platforms. For custom boards all the Tegra MACs can support this profile.

Available in all Standard (non-Safety), Safety and Safety overlay platform configurations.

IEEE 802.1AS Command

```

// #Server Mode
sudo ./ptp4l -f ./gPTP.cfg -p /dev/ptp<X> -i <iface> -m -17

// #client Mode

sudo ./ptp4l -f ./gPTP_slave.cfg -p /dev/ptp<X> -i <iface> -m -17

// where:// <iface> : Interface name i.e eqos_0, mgbe0_0, mgbe1_0 on which PTP is
// supposed to be started. It could be a VLAN interface above the primary interface.// /
// dev/ptp<X> - PTP dev node associated with the interface. Use "ethtool -T" command to
// identify the hardware clock instance i.e <X>

```

PTP Bridging in Orin

Orin SOC has 5 Ethernet MAC controllers embedded in it. Out of which only one can act as PTP secondary interface to synchronize the Tegra with external GM. There could be

use-cases where the time info coming from external GM needs to be forwarded to the devices connected to other mac controllers. This requires a PTP time aware bridge like implementation in Tegra. Starting 6.0.2 a new feature is included in lower layer to sync multiple MAC PHCs from an external GM. As of now, support is there for only 2 MACs in a given configuration.

Primary (source) and secondary (sink) MAC selection is configurable through device tree. Check the device tree binding doc in Documentation/devicetree/bindings/platform/tegra/tegra-nvethernet.txt.



Note: By default, the MGBE2→ EQOS sync is enabled on boot-up starting 6.0.2 release.

PTP-TSC Sync

Orin SOC provides a hardware assisted mechanism to align-sync the TSC (clock/counter) with one of the Tegra SOC Ethernet MAC on second boundary using the PPS signal from the MAC. This feature is useful in synchronizing camera fsync signals (used for camera frame capture/timestamping) with PTP time in order to reduce the timestamp jitter between sensors timestamped with PTP domain. This feature is also useful for co-relation of TSC and PTP time-domain timestamps (needed for sensor fusion application) as it ensures a fixed offset (no run-time drift) between PTP and TSC counters over the time. The MAC selection for TSC reference is based on the PTP topology of the platform. It must be always the MAC acting as PTP client for External GTM directly or through switch.

PTP time source MAC selection is configurable through NvTime2 modules configuration section.



Note: By default, the MGBE2→ TSC locking enabled on boot starting 6.0.3 release.

NvPPS

NvPPS module is Linux kernel module/driver which provide a single point of access to all the user applications to query:

- > Current synchronized PTP time from PHC
- > Event co-related PTP and TSC time domain timestamps

This module is enabled by default in the DRIVE OS kernel.

Operating Mode

NvPPS module has two operating modes.

1. **GPIO/PPS mode :** The NvPPS driver takes external 1PPS signal as input from devices like GPS etc.. on Tegra GPIO pin as an event and reads the timestamps from relevant time domains for correlation.
2. **Timer mode:** In this mode event for co-relation is based on periodic SW timer scheduled inside NvPPS module.

Timer mode is useful when there is no external device like GPS present for 1 PPS signal generation in system design or if there is no provision in platform (e.g P3663) to receive PPS signal through Tegra GPIO.

Userspace exposed interfaces

The module exposes a dev node named as "/dev/nvpps0". The dev node supports following IOCTL calls:

SN	IOCTL	Description
1	NVPPS_GETTIMESTAMP	Get current timestamps from kernel and PTP time from primary PHC interface
2	NVPPS_GETEVENT	Get correlated timestamps of different time-domains (TSC, PTP etc.) at the last event.
3	NVPPS_GETVERSION	Get NvPPS driver and API versions.
4	NVPPS_GETPARAMS	<p>Get the current NVPPS parameters. Like Operating mode(in GPIO or Timer mode) and TSC timestamp mode(in Nanosec or Counter mode).</p> <p>For the mode values to be used, please refer ioctl header file mentioned below.</p>
5	NVPPS_SETPARAMS	<p>Set the NvPPS parameters, described in the NVPPS_GETPARAMS section. This IOCTL can be used to update NvPPS parameters in runtime.</p> <p>Call this IOCTL with this command. Update configuration (operating mode and TSC timestamp mode) are required to set the parameters.</p> <p>The parameter settings in NvPPS module are common to all client applications.</p>

The data structures required for supporting the IOCTLs are defined in <top>/drive-linux/kernel/source/oss_src/nvidia-oot/include/uapi/linux/nvpps_ioctl.h.

Kernel space exposed interface

In addition to User space interfaces, NvPPS module exports a kernel API for other kernel modules to get the current synchronized PTP time in kernel. Refer to <top>/drive-linux/kernel/source/oss_src/nvidia-oot/include/linux/nvpps.h.

```
/*
 * Get PTP time
 * Clients may call this API whenever PTP time is needed.
 * If PTP time source is not registered, returns -EINVAL
 *
 * This API is available irrespective of nvpps dt availability
 * When nvpps dt node is not present, interface name will
 * default to "eth0".
 */
int nvpps_get_ptp_ts(void *ts)
```

NvPPS module configuration

Following static configuration is possible through NvPPS Device tree node. Refer to device tree binding doc <top>/drive-linux/kernel/source/oss_src/nvidia-oot/Documentation/devicetree/bindings/nvpps/nvpps.txt.

- > Primary PTP interface (prop_name: "interface") - The MAC interface which connects to external GM.
- > Secondary PTP interface (prop_name: "sec_interface") - The MAC which synchronizes with primary PTP MAC interface using PTP bridging.
- > 1PPS input pin (prop_name: "gpios") - Specifies GPIO pin detail on which 1PPS signal is available. Please refer platform specific NvPPS DT node more info.
- > PTP_TSC sync disable (prop name: "ptp_tsc_sync_dis") - Define this dt property to disable TSC syncing with PTP
- > PTP_TSC sync K_INT value (prop name: "ptp_tsc_k_int") - The property is used to define the K_INT value, which is used when calculating the delta value to apply, when using fast convergence algorithm. Minimum val: 0x0, Max val: 0xff, and default val: 0x70.
- > PTP_TSC sync LOCK_THRESHOLD value (prop name: "ptp_tsc_lock_threshold") - The property is used to define the LOCK_THRESHOLD value, The PTP_TSC lock is deemed lost when the absolute diff value exceeds this threshold value. The reset value 0x1F correspond to 1us. Min: 0x1F(1us), max:0xFFFF(approx 2.1ms), Default: 0x26c(20us). eg: if 50us needs to be programmed, then 0x1F * 0x32(50) = 0x60E

3.8.8 TSN API

TSN is grouped into three basic key component categories required for complete real-time communication. Multiple specifications fall under these three categories.

- > Time Synchronization i.e. 802.1AS(gPTP)

- > Scheduling and traffic shaping - i.e 802.1Qav, 802.1Qbv, 802.1Qbu/802.3br, 802.1CB and 802.1Qci.
- > Selection of communication paths, reservation and fault tolerance - i.e SRP (802.1Qat) and 802.1Qcc are key standards that define the system configuration of TSN networks.

Standards	Support on Orin Software and Hardware
802.1AS	YES
802.1 Qbu + 802.3br (egress path)	YES
802.1 Qbv (egress path)	YES
802.1 Qav (CBS) (egress path)	YES
IEEE 802.1Qat (SRP)	NO

This document provides sample application code for reference along with documenting the ioctl interface, which can be used to write customized applications to validate TSN specifications for Linux and QNX operating systems. Sample applications are used to configure end-to-end connections between two Ethernet interfaces

Terminology

TC - Traffic class value from 0 to 7.

GCL - gate control list, this is reliant to 802.1Qbv Specification

RQ - Frame Preemption Residue Queue. The Rx queue number to which the residual

preemption frames must be forwarded. Preemption frames that are tagged and pass the SA/DA/VLAN filtering are routed and All other frames are treated as residual frames and are routed to the queue number mentioned in this field. The Queue-0 is used as a default queue for express frames, so this field cannot be programmed to a value 0.

APP - application

OS - Operating system

DUT - Target device

Linux OS

This section is explaining how to use driver provided private IOCTL for configuring hardware in Linux OS.

Using IOCTL Interfaces

```

/* include */
#include "ether_export.h"

static int open_socket(char *ifname)
{
    int sockfd;

    sockfd = socket(PF_INET, SOCK_STREAM, 0);
    return sockfd;
}

static void close_socket(int sockfd)
{
    close(sockfd);
}

int main(int argc, char *argv[])
{
    int sockfd;
    static char *if_name;
    struct ifreq ifr;
    struct ifr_data_struct data;
    struct osi_est_config cfg;

    sockfd = open_socket(argv[1]);
    if_name = strdup(argv[1]);

    /* Update all fields of osi_est_config from user input for example...*/
    cfg.en_dis = atoi(argv1);
    cfg.btr[0] = atoi(argv2);
    cfg.btr[1] = atoi(argv3);
    cfg.btr_offset[0] = atoi(argv4);
    cfg.btr_offset[1] = atoi(argv5);
    cfg.ctr[0] = strtoll(argv6, NULL, 16);
    cfg.ctr[1] = strtoll(argv7, NULL, 16);

    data.cmd = ETHER_EST_CFG;
    data.ptr = &cfg;
    strcpy(ifr.ifr_ifrn.ifrn_name, ifname);
    ifr.ifr_ifru.ifru_data = &data;

    ret = ioctl(sockfd, DWC_ETH_QOS_PRV_IOCTL, &ifr);
    if (ret < 0)
        printf("IOCTL Error in %s()\n", __func__);
    else
        printf("Configured successfully\n");
    close_socket(sockfd);
    Return ret;
}

```

IOCTL Supported Commands

```
/** @defgroup private IOCTL related info
 * @brief MACRO are defined for driver supported
 * private IOCTLS. These IOCTLS can be called using
 * SIOCDEVPRIVATE custom ioctl command.
 */
/** Line speed */
#define EQOS_GET_CONNECTED_SPEED      25
/** To set HW AVB configuration from user application */
#define ETHER_AVB_ALGORITHM          27
/** To get current configuration in HW */
#define ETHER_GET_AVB_ALGORITHM       46
/** To configure EST(802.1 bv) in HW */
#define ETHER_CONFIG_EST              49
/** For configure FPE (802.1 bu + 803.2 br) in HW */
#define ETHER_CONFIG_FPE              50
/** @} */
```

Data Structures

Data structure header files are in the `drive-linux/include` folder.

Sample Applications

The sample applications use development utility code and should not be used for production.

802.1Qbu + 802.3br

IEEE 802.1 Qbu stops the transmission of long, non-critical frames to prioritize time-sensitive traffic, addressing the problem of transmission hogging. A major challenge for the timely transfer of critical messages is the presence of legacy traffic sharing the same network. Once a packet travels down a wire, it will block the wire from other packets until the end of the packet is reached.

To counter this issue, IEEE defined two standards- IEEE 802.1Qbu and IEEE802.3br, to support preemption. These standards, which build upon the TAS feature in 802.1Qbv, allow devices to preempt the transmission of non-TSN Ethernet frames (often legacy traffic) to prioritize high priority frames, while allowing the remainder of the interrupted frame to be sent later.

Syntax

```
.nvether_sample_app <interface name> fpe <TC preemption mask> <RQ>
```

Parameters

- > `<TC preemption mask>` - preemption- bit corresponding to TC should have value 1 else express/default bit value 0.

- > RQ - Residual/default queue for unfiltered preemptable packets. This value can be any enabled Rx queue except rx queue 0.

Stats: ethtool -S <interface name>

- > mmc_tx_fpe_frag_cnt - Tx fragment count
- > mmc_tx_fpe_hold_req_cnt - Tx set-hold count
- > mmc_rx_packet_reass_err_cnt - Rx reassembly error
- > mmc_rx_packet_smd_err_cnt - Rx SMD error
- > mmc_rx_packet_asm_ok_cnt - Rx assembly ok
- > mmc_rx_fpe_fragment_cnt - Rx fragment count.



Note:

- > Rx Queue 0 can not be RQ
- > Please use ethtool -S <interface name> to get stats related to FPE
- > Use the primary interface i.e eqos_0, mgbe2_0, etc. for configuration
- > FPE and MACSEC don't coexist incase of MGBE HW due to HW limitation.
- > FPE is configured when peer device supports FPE and responds to verify packet(SMD-V).

Example

- > To set preemption for TC 1 and TC 2 - (bit value representation rep 0x6), and RQ 2
 - nvether_sample_app <interface name> fpe 0x6 2
- > To disable TX preemption
 - nvether_sample_app <interface name> fpe 0x0 2

802.1Qbv

Traffic scheduling allows for different traffic classes to coexist with competing priorities on the same network. IEEE 802.1Qbv and IEEE 802.1Qbu work together to help manage this coexistence. IEEE 802.1 Qbv and IEEE 802.1Qbu can prioritize different traffic classes and enable time-sensitive data.

Devices/systems with IEEE 802.1Qbv can prioritize TSN Ethernet frames on a schedule, while non-TSN Ethernet frames to be transmitted on a best-effort basis around the TSN frames. The 802.1Qbv standard defines up to eight queues per port for forwarding traffic where each frame is assigned to a queue based on a QoS priority. To control the flow of queued traffic to a TSN enabled switch, this standard defines a time-aware shaper (TAS) mechanism that moderates queue traffic, preventing delays during scheduled transmission. Put simply, a gate in front of each queue opens at a specific point in time for time-sensitive traffic over standard (non-TSN) Ethernet packets. PTP should be initialized to make TSN working on DUT.

Syntax

```
nvether_sample_app <interface name> est <enable/disable> <base-time> <base-time offset>
<cycle time> <number of GCL entry> <GCL-entries>
```

Parameters

- > enable - 1 and disable - 0
- > base-time <nsec> <sec> put 0 0 if not sure - Value in decimal. If both values are 0 Driver will get PTP current time from MAC HW registers and add the base time offset into it.
- > base-time offset <nsec> <sec> - Value decimal. This offset is given to start a new GCL from base time value + offset.
- > cycle time <nsec> <sec> - Value in HEX. - ONE GCL full cycle time.
- > number of GCL entry max index 255 - value in decimal
- > GCL entries
 - [0x<8 bit gate mask> <24 bit interval - value in ns>]
 - [0x<8 bit gate mask> <24 bit interval - value in ns>]
 - :
 - :
 - [0x<8 bit gate mask> <24 bit interval in ns>]

**Note:**

- > When EST and FPE are enabled, bit 0 of the gate mask represents set-and-hold. In this case preemption is enabled for TC 0.
- > When only EST is enabled, all bits of the gate mask represent open-and-close. In this case all TC are expressed TC.
- > Disable tx flow control, such as ethtool -A <interface name> tx off rx off autoneg off
- > Use the primary interface, such as eqos_0, mgbe2_0 to configuration

Stats: ethtool -S <interface name>

- > const_gate_ctr_err - CGCE error count
- > head_of_line_blk_sch - Head of line block due to scheduling count
- > hlbs_q[0] - Head of line block due to scheduling count for TC 0
- > hlbs_q[1] - Head of line block due to scheduling count for TC 1
- > hlbs_q[2] - Head of line block due to scheduling count for TC 2
- > hlbs_q[3] - Head of line block due to scheduling count for TC 3
- > Hlbs_q[4] - Head of line block due to scheduling count for TC 4
- > hlbs_q[5] - Head of line block due to scheduling count for TC 5
- > hlbs_q[6] - Head of line block due to scheduling count for TC 6
- > hlbs_q[7] - Head of line block due to scheduling count for TC 7
- > head_of_line_blk_frm - Head of line block due to size count
- > hlfq[0] - Head of line block due to size count for TCO
- > hlfq[1] - Head of line block due to size count for TC1
- > hlfq[2] - Head of line block due to size count for TC2
- > hlfq[3] - Head of line block due to size count for TC3
- > hlfq[4] - Head of line block due to size count for TC4
- > hlfq[5] - Head of line block due to size count for TC5
- > hlfq[6] - Head of line block due to size count for TC6

- > hlf_q[7] - Head of line block due to size count for TC7
- > base_time_reg_err - Base time wrongly programmed in the past.
- > sw_own_list_complete - GCL switch happens successfully. This will increase this counter by 1.

Example

- > Enable EST with cycle time 16,777,685 ns, 2 GCL entries where in 1st entry for TCO open for 470 ns and in 2nd entry no gate open for 16,777,685 ns.
 - nvether_sample_app eth1 est 1 0 0 0 10 0x10001d5 0 2 0x10001d6 0xffffffff
- > Example for FPE and EST enable with set-and-hold feature
 - Configure FPE as mentioned in 802.1Qbu + 802.3br
 - Configure EST with set-and-hold feature like for case “Enable EST with cycle time 16,777,685 ns, 2 GCL entries where in 1st entry for TCO open for 470 ns with set-and-hold bit set and in 2nd entry no gate open for 16,777,215 ns”

```
nvether_sample_app eth1 est 1 0 0 0 10 0x10001d5 0 2 0x30001d6 0xffffffff
```

Check if the New GCL is Activated

GCL configuration and GCL activation are different. Note that programmed GCL is activated based on base time programmed. Check the results of ethtool -S <interface name> to check if the sw_own_list_complete value updated from the last value or if an error counter increased.

802.1Qav

Originally developed to meet the bandwidth requirements of audio/video streams while preserving bandwidth for best effort traffic, IEEE 802.1Qav, known as Forwarding and Queueing for Time-Sensitive Streams (FQTSS) and later integrated into 802.1Q, defines a credit-based traffic shaper for reserving bandwidth for A/V streams in a bridged network.

In bridges, reserved A/V stream traffic is prioritized over best effort traffic as long as sufficient credits are available to transmit the higher-priority A/V data. As with end stations sourcing stream data, the credit-based traffic shaper results in stream data distributed evenly across a bridged network over time

Syntax

```
nvether_sample_app <interface name> avb <qinx> <algorithm_value> <bw> <credit_control>
<tcinx>
```

Parameters

- > qinx - Tx Queue index
- > algorithm value
 - 0 - Default (strict priority for EQOS and EST for MGBE)
 - 1 - CBS (credit based shaper)

- > bw - Value in terms of percentage of bandwidth to be allocated (1-100)
- > credit_control
 - 0 - disabled credit_control. The credit accumulates even when there is no packet waiting in TC<n> and another TC is transmitting.
 - 1 - enabled credit_control. When there is no packet waiting in TC<n> and other TC is transmitting, no credit is accumulated.
- > tcinx - TC mapping for <qinx>

**Note:**

- > credit_control is applicable for cbs only. CBS configuration is per TC.
- > Use the primary interface, such as eqos_0, mgbe2_0 for configuration.

Example

- > • Give 20 % bandwidth to TxQ/TC 2.

```
nvether_sample_app <interface name> avb 2 1 20 1 2
```

- Reset to default tx selection algorithm for TxQ/TC 2

```
nvether_sample_app <interface name> avb 2 0 20 1 2
```

VLAN(802.1Q)

In Linux, to create VLAN and MAP the ingress and egress priority to SKB priority following Linux standard tools and utilities can be used. Example commands are as follows:

```
ip link add link eth0 name eth0.5 type vlan id 5
ip addr add 10.0.1.6/24 brd 10.0.1.255 dev eth0.5
ip link set dev eth0.5 up

ip link add link eth0 name eth0.5 type vlan id 5 egress 0:1 1:2 4:4
```

In the previous command, the internal skb prio 0 is mapped to VLAN prio 1, skb prio 1 to VLAN prio 2 and 4 to 4.

You can also use in built application vconfig such as vconfig set_egress_map [vlan-name] [skb_priority] [vlan_qos]

PTP (802.1AS) - 2 step PTP

Nodes in the TSN network communicate with each other in real time and need a shared understanding of time to agree on corrective actions, recognize each other's state, and cooperate together.

The IEEE 802.1AS project created a profile of the IEEE 1588 PTP synchronization protocol for TSN. This profile will enable clock synchronization compatibility between different TSN devices. 802.1AS also addresses support for fault tolerance and multiple active synchronization masters.

3.8.9 FRP (Flexible Receive Parser) Validation

This section explains how to enable an in-built programmable parser to parse the received Ethernet packet and provide an interface for users to configure rules in the parser.

The sample(QM) application shows how to configure parser rules for user-defined actions matched Rx packets.

FRP (Flexible Receive Parser)

The MAC hardware controller has an in-built programmable parser to parse the Ethernet packet based on a software-controlled/programmable rule-set. This supports filtering and packet steering decisions (DMA channel selection) of the received packet based on any header field, 64/124 bytes from SOF (start of frame), of existing protocols or custom and future protocols. The parser uses a software programmed lookup table that contains instructions and filtering and routing decisions taken by the controller on the received packets.



Note: When FRP-based routing is enabled, MAC, RSS and IP based routing will not apply. Create FRP rules for any such added routing rules.

Scope

There is a designated interface for FRP configuration, such as default 0, and this dedicated instance can configure any physical MAC controller instance of that hardware.

Abbreviations

- > FRP -Flexible Receive Parser
- > MAC - Medium access control
- > MAC - Medium access control
- > RSS - Receive side scaling
- > IP - Internet protocol
- > DUT - Device under test
- > OK index(OKI)
- > NIC - Next Instruction Control.

```
if (NIC==0) continue parse from OK Index
else /*(NIC ==1)*/ continue sequentially (parse the next entry in the instruction
table)
```

- > L2 Filter - Layer 2 MAC address filter

Linux OS



Note: The sample application is for development purposes and not for production.

Linux nvether_sample_app utility

QM sample application binary for Nvidia DUT (nvether_sample_app)

- > Part of SDK release for reference

To check the version:

```
./nvether_sample_app
```

Look for string such as Version(4), which means 4.

FRP_ADD Command Syntax

AV+L:

```
nvether_sample_app <primary interface> frp_add <ID> <Match> <Type> <Filter Mode>
<Offset> <OKI> <DMASel>
```

Parameters

- > **Id** - FRP table ID to add (0 to 255)
- > **Match** - Match data is used for comparing
 - MAX 12 bytes data
- > **Type** - Match data type
 - 0 - Normal data
 - 1 - L2 DA MAC
 - 2 - L2 SA MAC
 - 3 - L3 Source IP
 - 4 - L3 Destination IP
 - 5 - L4 UDP Source Port
 - 6 - L4 UDP Destination Port
 - 7 - L4 TCP Source Port
 - 8 - L4 TCP Destination Port
 - 9 - VLAN Tag
- > **Mode** - Filter mode for the entry
 - 0 - Accept and route
 - 1 - Reject and Drop
 - 2 - Accept and Bypass FRP Route
 - 3 - Link to OK Index
 - 4 - Inverse the Match, Accept and route
 - 5 - Inverse the Match, Reject and Drop
 - 6 - Inverse the Match, Accept and Bypass FRP Route
 - 7 - Inverse the Match, Link to OK Index
- > **Offset** - Frame offset of Match data
- > **OKI** - When NIC set give the value for Next Instruction
- > **DMASel** - Bit selection of DMA channels to route the frame
 - Bit[0] - DMA channel 0
 - ...
 - Bit [N] - DMA channel N]

- Max N for EQOS HW is 7
- Max N for MGBE is 9



Note: Packet duplication to multiple DMA channels works only for MC/BC packets and Highest RxQ needs to be enabled on the DT. Highest queue (Rx queue 9 for MGBE and Rx queue 7 for Orin platform)

Example

1. Add FRP rule to accept and route packets from 192.168.1.100

```
nvether_sample_app <primary interface> frp_add 0 C0A80164 4 0 0 0 1
```

2. Add FRP rule to reject and drop packets from 192.168.1.100

```
nvether_sample_app <primary interface> frp_add 0 C0A80164 4 1 0 0 1
```

FRP_UPDATE Command Syntax

AV+L:

```
nvether_sample_app <primary interface> frp_update <ID> <Match> <Type> <Filter Mode> <Offset> <OKI> <DMASel>
```

- > **Id** - FRP table ID to add (0 to 255)
- > **Match** - Match data is used for comparing
 - MAX 12 bytes data
- > **Type** - Match data type
 - 0 - Normal data
 - 1 - L2 DA MAC
 - 2 - L2 SA MAC
 - 3 - L3 Source IP
 - 4 - L3 Destination IP
 - 5 - L4 UDP Source Port
 - 6 - L4 UDP Destination Port
 - 7 - L4 TCP Source Port
 - 8 - L4 TCP Destination Port
 - 9 - VLAN Tag
- > **Mode** - Filter mode for the entry
 - 0 - Accept and route
 - 1 - Reject and Drop
 - 2 - Accept and Bypass FRP Route
 - 3 - Link to OK Index
 - 4 - Inverse the Match, Accept and route
 - 5 - Inverse the Match, Reject and Drop
 - 6 - Inverse the Match, Accept and Bypass FRP Route
 - 7 - Inverse the Match, Link to OK Index

- > **Offset** - Frame offset of Match data
- > **OKI** - When NIC set give the value for Next Instruction
- > **DMASel** - Bit selection of DMA channels to route the frame
 - Bit[0] - DMA channel 0
 - ...
 - Bit [N] - DMA channel N]
 - Max N for EQOS HW is 7
 - Max N for MGBE is 9



Note: The Multiple DMA channel selection only works for MC/BC packets and Highest RxQ needs to be enabled on the DT. Highest queue (9 for MGBE and 7 for Orin platform)

Example

1. Add FRP rule to accept and route packets from 192.168.1.100

```
nvether_sample_app <primary interface> frp_add 0 C0A80164 4 0 0 0 1
```

2. Update FRP rule to reject and drop packets from 192.168.1.100

```
nvether_sample_app <primary interface> frp_update 0 C0A80164 4 1 0 0 1
```

FRP_DEL Command Syntax

AV+L

```
./nvether_sample_app <primary interface> frp_del <ID>
```

Parameters

- > Id - FRP table ID to add (0 to 255)

Example

- > Delete FRP rule at ID 0.

```
nvether_sample_app <primary interface> frp_del 0
```

Additional Example Syntax

L2 DA accept Filtering and DMA route

- > Delete Old FRP entries with frp_del command

```
nvether_sample_app <primary interface> frp_del 0
```

- > Add FRP rule to ADD DA f2:3b:00:06:87:ff:

```
nvether_sample_app <primary interface> frp_add 0 f23b000687ff 1 0 0 0 0x10
```

L2 DA Reject Filtering and DMA route

- > Add FRP rule to ADD DA f2:3b:00:06:87:ff:

```
nvether_sample_app <primary interface> frp_update 0 f23b000687ff 1 1 0 0 0x10
```

L2 MC DA Accept Filtering and Multiple DMA Channels Route

- > DUT side Add/Update FRP rule to MC MAC 01:00:5E:01:01:01

```
nvether_sample_app <primary interface> frp_update 0 01005E010101 1 0 0 0 0x3FF
```

L2 MC DA Reject Filtering

- > DUT side Add/Update FRP rule to MC MAC 01:00:5E:01:01:01

```
nvether_sample_app <primary interface> frp_update 0 01005E010101 1 1 0 0 0x3FF
```

L2 BC DA Accept Filtering and Multiple DMA Channels

- > DUT side Add/Update FRP rule for BC MAC FF:FF:FF:FF:FF:FF

```
nvether_sample_app <primary interface> frp_update 0 FFFFFFFFFFFF 1 0 0 0 0x3FF
```

L2 BC DA Reject Filtering

- > DUT side Add/Update FRP rule for BC MAC FF:FF:FF:FF:FF:FF

```
nvether_sample_app <primary interface> frp_update 0 FFFFFFFFFF 1 1 0 0 0x3FF
```

L2 SA Reject filtering

- > DUT side Add/Update FRP table 0 and 1 entry to ADD SA <<00:17:b6:00:00:00>>

```
nvether_sample_app <primary interface> frp_update 0 <<0017b6000000>> 2 1 0 0 0x8
```

L3 SA IP Reject filtering

- > DUT Side Add/Update FRP table entry 0 and 1 for Source IP 192.168.1.3

```
nvether_sample_app <primary interface> frp_update 0 C0A80103 3 1 0 0 0x2
```

Enable VLAN accept filtering using FRP command

```
nvether_sample_app <primary interface> frp_update 0 0005 9 0 0 0 0x2
```

Enable VLAN reject filtering FRP commands

```
nvether_sample_app <primary interface> frp_update 0 0005 9 1 0 0 0x2
```

L2 DA + L2 SA accept Filtering and DMA route

- > Add FRP rule for L2 DA <<f2:3b:00:06:87:ff>> L2 SA and link both rules:

- nvether_sample_app <primary interface> frp_add 0 <f23b000687ff> 1 0 0 0 0x10
- nvether_sample_app <primary interface> frp_add 1 <f23b000687ff> 2 0 0 0 0x10
- nvether_sample_app <primary interface> frp_update 0 <f23b000687ff> 1 4 0 0 0x10

L2 DA + L2 SA Reject Filtering

- > Add FRP rule for L2 DA <<f2:3b:00:06:87:ff>> L2 SA and link both rules:

- nvether_sample_app <primary interface> frp_add 0 <f23b000687ff> 1 0 0 0 0x10
- nvether_sample_app <primary interface> frp_add 1 <f23b000687ff> 2 1 0 0 0x10

- nvether_sample_app <primary interface> frp_update 0 <f23b000687ff> 1 4 0 0
0x10

L2 Address Filter Command Syntax

AV+L

```
nvether_sample_app <interface name> l2_filter <filter_no> <enable/disable> <mac addr>
                  Filter_no
Index 0 to 31.
enable/disable
0 - to disable filter, 1 - to enable filter\n"
mac addr - MAC address(Ex - 94:18:82:71:ae:1d)]\n"
```


Note:

- > Rx Packets are routed to the DMA channel, which is bound to the interface used for configuration.
- > This feature can be used only with Ethernet virtualization enabled.
- > As there are multiple default L2 addresses configured, use index numbers in reverse order reverse, such as 31, 30, 29, to avoid overwriting.

Example

- > To add/enable AA:10:18:2b:8f:8a at index 9
 - eqos_0 l2_filter 9 1 AA:10:18:2b:8f:8a
- > To delete/disable AA:10:18:2b:8f:8a at index 9
 - eqos_0 l2_filter 9 0 AA:10:18:2b:8f:8a

3.8.10 Marvell Switch Firmware Management

The following table describes Marvell switch current firmware versions on the switches.:

Platform	Switch	Version	Firmware Name	Function supported
P3663	Oak/88Q5072	0.07.1186.01	P3663_88Q5072_fla	AVNU CDS 1.6 PTP enabled as per role described in PTP section
P3710	Oak/88Q5072	0.07.1186.01	P3663_88Q5072_fla	AVNU CDS 1.6 PTP enabled as per role described in PTP section
	Spruce/88Q6113	0.07.1186.01	P3710_88Q6113_fla	AVNU CDS 1.6 PTP enabled as per role described in PTP section

Switch firmware update from Orin over Ethernet

Starting in 6.0.0.1, it's possible to update the Marvell switch firmware from Orin through "update_firmware.sh" script packaged in file system. Silent feature of update process/framework is as below:

- > Automatic upgrade is by default enabled in AV+L standard build only through boot-up script.
- > Manual force upgrade/downgrade is also possible via scripts in AV+L, AV+Q & AV+Q+Q standard builds.
- > Provision to get the current flashed firmware is also possible via script.



Note: Aurix MCU must be pre-flashed with firmware version >= 6.0.0.1.

Manual up-grade/de-grade the firmware

For manual up-grade/de-grade, different scripts are provided per switch. Firmware binaries to be used for this purpose can be found in `/lib/firmware/marvell_ethernet`. Following are the command references and their locations.

```
//P3663 - Oak
$ sudo /bin/bash /lib/firmware/marvell_ethernet/driveota/P3663_88Q5072.sh --Install <FW>

//P3710 - Oak
$ sudo /bin/bash /lib/firmware/marvell_ethernet/driveota/P3710_88Q5072.sh --Install <FW>

//P3710 - Spruce
$ sudo /bin/bash /lib/firmware/marvell_ethernet/driveota/P3710_88Q6113.sh --Install <FW>
```

Print current flashed firmware version:

```
//P3663 - Oak
$ sudo /bin/bash /lib/firmware/marvell_ethernet/driveota/P3663_88Q5072.sh --
GetCurrentVersion

//P3710 - Oak
$ sudo /bin/bash /lib/firmware/marvell_ethernet/driveota/P3710_88Q5072.sh --
GetCurrentVersion

//P3710 - Spruce
```

```
$sudo /bin/bash /lib/firmware/marvell_ethernet/driveota/P3710_88Q6113.sh --
GetCurrentVersion
```

**Note:**

1. Newly installed firmware will take effect only after aurix is power cycled (As aurix resets the switch)
2. It is not recommended to update the switch firmware manually unless absolutely required or informed by NVIDIA
3. It's advised to run the script intended for a specific board on that board only(ex: Run P3710_88Q6113.sh on P3710 only) as the recovery from firmware corruption cannot be done without manually flashing using JATG
4. File format of the firmware binary to be provided as input to "-Install" is fixed. Please refer SWITCH_FW_FORMAT from <board>_<switch>.sh (ex:P3663_88Q5072.sh). Binary provided in any other format will be rejected.
5. Output of "GetCurrentVersion" follows following format.

x.xx.xxxx.xx

0.07.1186.01

3.8.11 P3898 Switch Firmware Management

The following table describes p3898 switch firmware versions:

Platform	Version	Firmware Name	Function Supported
P3898	5.1.1	P3898-switch-fw.v5.01.01.bin	AVNU CDS 1.6 PTP enabled as per role described in PTP Section

Switch Firmware Update from Orin

It's possible to update the p3898 switch firmware from Orin through the nv_3898_switch_ota.sh script packaged in the filesystem. The salient feature of update process is as below:

Feature	Description	Invocation
Auto Update	Same as above. And if update required, then also update Switch FW	sudo /bin/bash /etc/systemd/scripts/nv_3898_switch_ota.sh --autoupdate

Feature	Description	Invocation
Check Update	Check if update is required for Switch FW by comparing flashed FW version with Switch FW present in /lib/firmware/3898_ethernet/	<code>sudo /bin/bash /etc/systemd/scripts/nv_3898_switch_ota.sh --checkupdate</code>
Force Update	Accepts FW filename as argument and do forceful update of Switch FW	<code>sudo /bin/bash /etc/systemd/scripts/nv_3898_switch_ota.sh --forceupdate <FW file name></code> For example: <code>sudo /bin/bash /etc/systemd/scripts/nv_3898_switch_ota.sh --forceupdate P3898-switch-fw.v5.01.01.bin</code>
Print Current Version	Print Current Switch FW version	<code>sudo /bin/bash /etc/systemd/scripts/nv_3898_switch_ota.sh --printversion</code>

Note the following:

1. Switch firmware files must be placed at path `/lib/firmware/3898_ethernet/`.
2. Switch firmware files must follow the naming convention `P3898-switch-fw vX.YY.ZZ.bin/img`, where X is the major number, YY is the minor number ,and ZZ is the revision number.
3. The p3898 switch OTA systemd service (`nv_3898_switch_ota.service`) currently checks for updates and prints the information on the console. Automatic update of switch firmware is not enabled as it takes more time (around 5 minutes).
4. User are required to update the switch firmware by using the Auto Update command listed above if an update is required.
5. Newly installed firmware only takes effect only after the board is power cycled.

3.8.12 PCIe Ethernet

PCIe Ethernet(LAN7431) is a 1G interface to be used only for development purpose like drive OTA update and others.

The interface is available on both P3663 and P3710.

Since its only used for development purposes, PTP support is not used/verified on this interface.

The LAN7431 interface enumerates as `enP7p1s0`

3.8.13 Marvel 88Q4364 PHY Firmware Update

Prerequisite: ensure the interface is up before you start firmware flash.

To flash the latest firmware version, go to the firmware flashing tool path:/lib/firmware/marvell_ethernet/88Q4364/

Flash, as follows:

```
./lib/firmware/marvell_ethernet/88Q4364/flash_4364 --install mgbe0_0 Arc-7.1.8/fw.image-  
ARC_9KB_nvidia_Main_MSMode-GPIO_ID58_VER2031.nvm.bin
```

Once flashed, check the version number:

```
./lib/firmware/marvell_ethernet/88Q4364/flash_4364 --GetCurrentVersion mgbe0_0
```

3.8.14 MACsec Overview

The IEEE 802.1AE MACsec standard and its amendments provide the infrastructure for secure L2 network communication, offering authentication and confidentiality of the data communicated between two peers on the network.

The NVIDIA MACsec(NvMACsec) implementation for Orin has two (2) major entities: KaY entity (IEEE 802.1x standard) and SecY entity (IEEE 802.1AE standard). In NvMACsec, KaY entity is implemented in nv_macsec_wpa_supplicant application and SecY entity is part of the MACsec hardware. The NvMACsec nv_macsec_wpa_supplicant tool implemented using open source wpa_supplicant version 2.10 and supports:

- > Authentication only, no confidentiality
- > Either GCM-AES-128 or GCM-AES-256 at a time
- > 32-bit Packet number and does not support Extended Packet Number (64-bit PN)

NvMACsec is enabled by default in the ethernet DT variable “nvidia,macsec-enable = <0x1>”. When enabled, ethernet MTU size is reduced by 34 bytes to accommodate MACsec related headers.

NvMACsec can be used on Orin by launching a supplicant process in the background using below command with root privileges. NvMACsec supplicant process must not be killed abruptly and can be gracefully terminated with SIGINT(Ctrl+C) signal. In QNX, the below command can be prepended with SOCK=<sock_variable> to run the supplicant application on a specific interface, the same can be applied to run the supplicant on Guest OS0 or Guest OS1:

```
nv_macsec_wpa_supplicant -i <interface_name> -D nv_macsec -c  
nv_wpa_supplicant_macsec.conf &
```

The example below adds launch of supplicant from bootup scripts in QNX.

```
eqos_1_09: eqos_1_09 {  
    cmd = "iolauncher -U  
    3333:3333,3660,3500,3350,3775,3640,2280,3000,2281,2282,2283,3780,3790,6025,6026,40002,40006,40007
```

```
--secpol-type supplicant_launch_t --set-var SOCK=/eqos_1 nv_macsec_wpa_supplicant -i
eqos_1 -D nv_macsec -c /etc/nv_wpa_supplicant_macsec_template.conf";
sc7 = "restart";
critical_process = "no";
heartbeat = "no";
oneshot = "no";
};
```

The contents of template conf file (nv_wpa_supplicant_macsec.conf):

```
eapol_version=3
ap_scan=0
network={
    key_mgmt=NONE
    eapol_flags=0
    mka_priority=16
    macsec_policy=1
    macsec_integ_only=1
    macsec_cak_len=16
    mka_cak_pkcs_id=MACSEC_CAK_eqos_0
    mka_ckn=112233445566778899AABBCCDDEEFF112233445566778899AABBCCDDEEFF1122
}
```

key_mgmt: Don't change the default value

eapol_flags: Don't change the default value

mka_priority : It is the MKA key server priority. Lower the value, higher is the key server priority.

macsec_policy: Flag to enable or disable MACsec. Make sure to set it to 1 for MACsec to be enabled.

macsec_integ_only: Flag to specify if encryption/integrity check to be enabled only. Orin doesn't support encryption. Make sure to set it to 1 for Nvmacsec.

macsec_cak_len: Length of the CAK in bytes programmed in secure storage.

mka_cak_pkcs_id: PKCS ID used while programming the CAK in secure storage. The NvMACsec will retrieve CAK handle from securely stored CAK using pkcs11 api's. Details on programing the CAK to secure storage can be referred from “Provisioning PKCS#11 Key Objects” section under “Understanding Security” chapter of PDK.

mka_ckn: Any 32 byte network name used while forming the SC.

macsec_cs_index: If the DUT is the key server this parameter is used to decide the cipher suite. This parameter 0 is configured to 0 to select GCM_AES_128, 1 to select GCM_AES_256

macsec_pn_exhaustion: If the DUT is the key server this parameter is used to program the number of frames after which Re-Keying needs to be initiated.

For debugging, NvMACsec driver supports the error counters and functional stats with applications. The stats can be obtained by issuing DEVCTLs with below details:

```
msg.i.dcmd= SI0CGDRVSPEC
```

`ifd.ifd_cmd = NV_MACSEC_DBG_CMD_READ_IRQ_STATS` macro in `nv_macsec.h` for error stats of MACSEC.

The following details MACsec IRQ counters:

```
/** Tx debug buffer capture done */
nveu64_t tx_dbg_capture_done;
/** Tx MTU check failed */
nveu64_t tx_mtu_check_fail;
/** Tx MAC CRC err */
nveu64_t tx_mac_crc_error;
/** Tx SC AN not valid */
nveu64_t tx_sc_an_not_valid;
/** Tx AES GCM buffer overflow */
nveu64_t tx_aes_gcm_buf_ovf;
/** Tx LUT lookup miss */
nveu64_t tx_lkup_miss;
/** Tx uninitialized key slot */
nveu64_t tx_uninit_key_slot;
/** Tx PN threshold reached */
nveu64_t tx_pn_threshold;
/** Tx PN exhausted */
nveu64_t tx_pn_exhausted;
/** Tx debug buffer capture done */
nveu64_t rx_dbg_capture_done;
/** Rx ICV error threshold */
nveu64_t rx_icv_err_threshold;
/** Rx replay error */
nveu64_t rx_replay_error;
/** Rx MTU check failed */
nveu64_t rx_mtu_check_fail;
/** Rx MAC CRC err */
nveu64_t rx_mac_crc_error;
/** Rx AES GCM buffer overflow */
nveu64_t rx_aes_gcm_buf_ovf;
/** Rx LUT lookup miss */
nveu64_t rx_lkup_miss;
/** Rx uninitialized key slot */
nveu64_t rx_uninit_key_slot;
/** Rx PN exhausted */
nveu64_t rx_pn_exhausted;
/** Secure reg violation */
nveu64_t secure_reg_viol;
```

`ifd.ifd_cmd = NV_MACSEC_DBG_CMD_READ_MM_CNTRS` macro in `nv_macsec.h` for functional stats of MACsec.

The following details MACsec MMC counters:

```
/** This counter provides the number of controller port macsec
 * untagged packets */
```

```

nveul64_t rx_pkts_no_tag;
/** This counter provides the number of controller port macsec
 * untaged packets validateFrame != strict */
nveul64_t rx_pkts_untagged;
/** This counter provides the number of invalid tag or icv packets */
nveul64_t rx_pkts_bad_tag;
/** This counter provides the number of no sc lookup hit or sc match
 * packets */
nveul64_t rx_pkts_no_sa_err;
/** This counter provides the number of no sc lookup hit or sc match
 * packets validateFrame != strict */
nveul64_t rx_pkts_no_sa;
/** This counter provides the number of late packets
 * received PN < lowest PN */
nveul64_t rx_pkts_late[16];
/** This counter provides the number of overrun packets */
nveul64_t rx_pkts_overrun;
/** This counter provides the number of octets after IVC passing */
nveul64_t rx_octets_validated;
/** This counter provides the number not valid packets */
nveul64_t rx_pkts_not_valid[16];
/** This counter provides the number of invalid packets */
nveul64_t in_pkts_invalid[16];
/** This counter provides the number of in packet delayed */
nveul64_t rx_pkts_delayed[16];
/** This counter provides the number of in packets un checked */
nveul64_t rx_pkts_unchecked[16];
/** This counter provides the number of in packets ok */
nveul64_t rx_pkts_ok[16];
/** This counter provides the number of out packets untaged */
nveul64_t tx_pkts_untagged;
/** This counter provides the number of out too long */
nveul64_t tx_pkts_too_long;
/** This counter provides the number of out packets protected */
nveul64_t tx_pkts_protected[16];
/** This counter provides the number of out octets protected */
nveul64_t tx_octets_protected;

```

If the MACSec Key is not already provisioned on the board but MACSec is enabled from the ethernet DT using "nvidia,macsec-enable", then expect the errors below to appear in boot logs. They not harmful if MACSec is not being used:

```

sessionSetup: C_FindObjects failed, 0x0
nvpkcs_session_setup: sessionSetup failed, 0x6
KaY-ieee802_1x_kay_create_mka: nvpkcs_session_setup() failed.

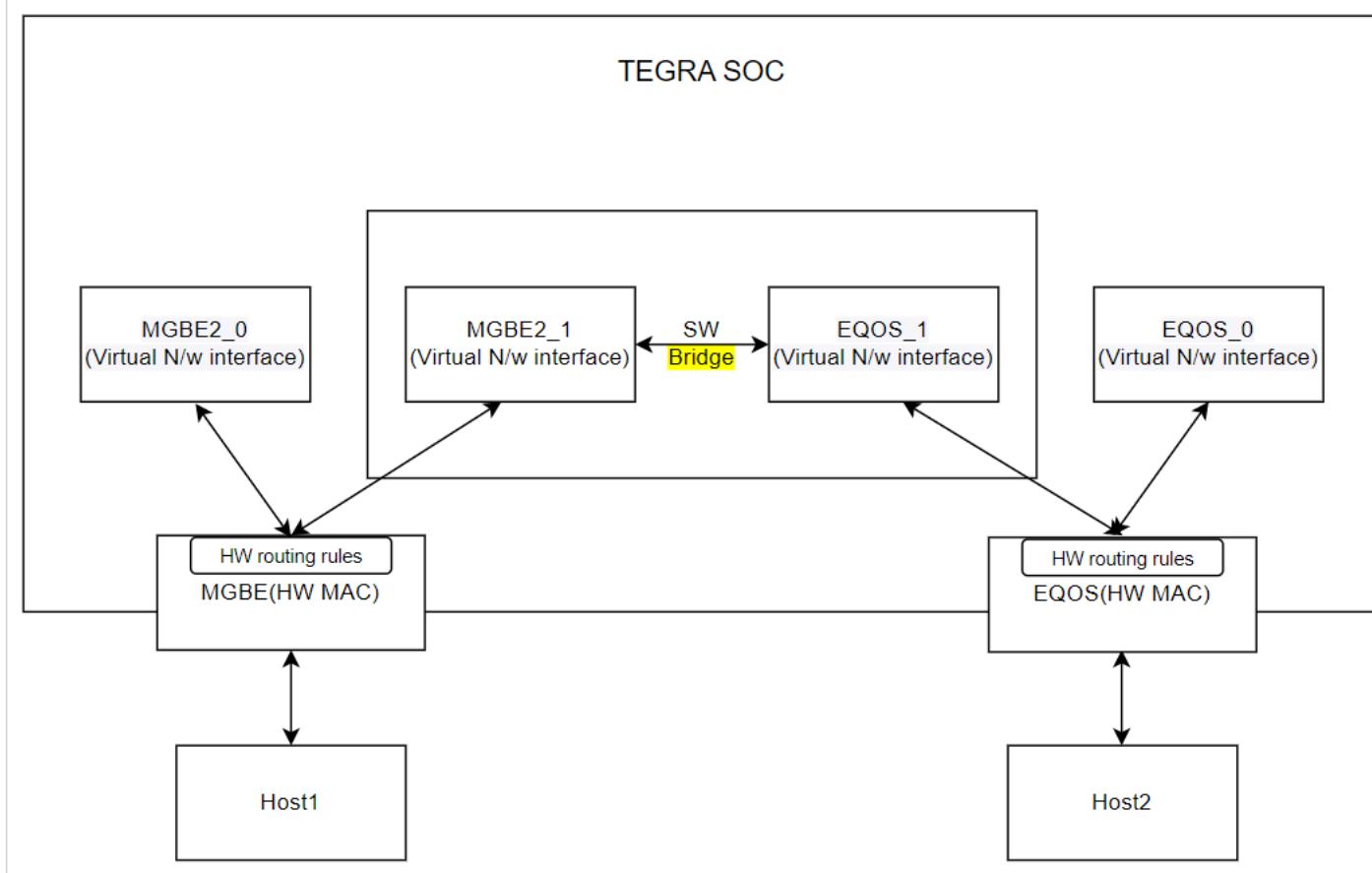
```

3.8.15 Layer-2 Bridge in Orin

DRIVE OS supports low bandwidth layer-2 network bridge between 2 or more Tegra ethernet MACs using ethernet MAC HW virtualization, ethernet MAC HW Flexible Receive Parser (FRP) features and brctl (from bridge-utils package in Linux).

The Layer-2 bridge is in "deny all" mode for all unicast & multicast packets. To allow packets through this bridge, FRP rules need to be programmed in the ethernet MAC.

A Layer-2 bridge is already deployed (without FRP rules) in NDAS usecase (starting 6.0.7.0) between MGBE2 & EQOS ethernet MAC. The block diagram is shown below.



Creating the Bridge

To create the bridge:

1. Virtualize the MAC interfaces using the steps mentioned in "Ethernet VF configuration" section under Networking
2. Enable creating of Layer 2 nw bridge by updating the device tree with device node '`ndas_nw_bridge`' and '`status = "okay"`'.
3. Create the Layer 2 bridge interface and add the interfaces using `brctl` (from `bridge-utils` package utility).
4. Assign zero IP address to the interfaces participating in bridge function.
5. Set up FRP rules to allow packets through the Layer-2 bridge. Refer below section for setting up FRP rules.

FRP Rules Setup Examples

This Layer-2 bridging solution requires combination of Layer-2 Destination Addr filtering rules and FRP rules in HW MACs participating in the bridging.

Prerequisites

- > • A sample tool named nvether_sample_app needs to be generated based on Networking header files packaged in include path of AV+L DRIVE OS SDK package, The tool is required to configure the FRP Rules and L2 filter rules.
- Following parameters need to be configured statically.
 - MAC address of devices connected across the layer-2 bridge.
 - Multicast addresses based on use case.

Refer to the Exported Networking element APIs in API reference section for the list of IOCTLs used by sample tool.

Example setup details:

Interface Name	MAC	IP
Host1 (host1_eth0)	b2:fc:eb:b3:f6:90	192.168.90.10
Host2 (host2_eth0)	8a:05:14:b6:23:01	192.168.100.10
MGBE2_0	7e:98:e2:75:ee:56	192.168.90.20
MGBE2_1	7e:98:e2:75:ee:57	192.168.90.40
EQOS_0	ca:46:a3:f7:fc:16	192.168.100.20
EQOS_1	ca:46:a3:f7:fc:17	192.168.100.40



Note: All MAC addresses in the table above are randomly generated.

Setup routing rules on Linux hosts

```
// On Host1
$ sudo ip route add 192.168.100.0/24 dev host1_eth0
//On Host2
$ sudo ip route add 192.168.90.0/24 dev host2_eth0
```

Example Use Cases

Add MAC L2 filter rule to allow packets to reach FRP engine of the ethernet MAC interface.

```
//Add "Host1"(connected to MGBE) MAC addr in EQOS MAC L2 filter
# ./nvether_sample_app eqos_0 l2_filter 10 1 b2:fc:eb:b3:f6:90

//Add "Host2"(connected to EQOS) MAC addr in MGBE2 MAC L2 filter
# ./nvether_sample_app mgbe2_0 l2_filter 10 1 8a:05:14:b6:23:01
```

Use case 1: Allow ping through the bridge using unicast MAC DA based filter.

```
//Examples of DA filter rules in EQOS & MGBE2

// FRP rules to set on EQOS interface
// Add FRP rule to filter, replicate and route Broadcast packets to both virtual n/w interfaces
./nvether_sample_app eqos_0 frp_add 0 ffffffff 1 0 0 0 0x5

// Add FRP rule to filter pkts with EQOS_1 DA MAC Addr and forward it to EQOS_1 virtual n/w
./nvether_sample_app eqos_0 frp_add 1 ca46a3f7fc17 1 0 0 0 0x4

// Add FRP rule to filter pkts with Host1 DA MAC addr and route them to EQOS_1 virtual n/w
./nvether_sample_app eqos_0 frp_add 2 b2fcebb3f690 1 0 0 0 0x4

// FRP rules to set on MGBE interface
// Add FRP rule to filter, replicate and route Broadcast packets to both virtual n/w interfaces
./nvether_sample_app mgbe2_0 frp_add 0 ffffffff 1 0 0 0 0x5

// Add FRP rule to filter pkts with MGBE2_1 DA MAC Addr and forward it to MGBE2_1 virtual n/w
./nvether_sample_app mgbe2_0 frp_add 1 7e98e275ee57 1 0 0 0 0x4

// Add FRP rule to filter pkts with Host2 DA MAC addr and route them to MGBE2_1 virtual n/w
./nvether_sample_app mgbe2_0 frp_add 2 8a0514b62301 1 0 0 0 0x4
```

Once the rules are set. Ping Host2 from Host1 and vice versa. Ping should pass.

Use case 2: Allow Multicast packets through the bridge using Multicast DA based filter.

```
//Multicast(MC) MAC addr => "01:00:5E:00:00:C8"
//AddPTP MC MAC addr in EQOS MAC L2 filter

#./nvether_sample_app eqos_0 l2_filter 10 1 01:00:5E:00:00:C8

//Add MC MAC addr in MGBE MAC L2 filter

#./nvether_sample_app mgbe2_0 l2_filter 10 1 01:00:5E:00:00:C8
```

```
// Examples of Multicast(MC) DA filter rules in EQOS & MGBE2

// FRP rules to set on EQOS interface
// Add FRP rule to filter, replicate and route Broadcast packets to both virtual n/w interfaces
./nvether_sample_app eqos_0 frp_add 0 ffffffffffffff 1 0 0 0 0x3

// Add FRP rule to filter pkts with EQOS_1 DA MAC Addr and forward it to EQOS_1 virtual n/w interface
./nvether_sample_app eqos_0 frp_add 1 ca46a3f7fc17 1 0 0 0 0x2

// Add FRP rule to filter pkts with MC DA MAC addr and route them to EQOS_1 virtual n/w interface
./nvether_sample_app eqos_0 frp_add 2 01005E0000C8 1 0 0 0 0x2

// FRP rules to set on MGBE interface
// Add FRP rule to filter, replicate and route Broadcast packets to both virtual n/w interfaces
./nvether_sample_app mgbe2_0 frp_add 0 ffffffffffffff 1 0 0 0 0x3

// Add FRP rule to filter pkts with MGBE2_1 DA MAC Addr and forward it to MGBE2_1 virtual n/w interface
./nvether_sample_app mgbe2_0 frp_add 1 7e98e275ee57 1 0 0 0 0x2

// Add FRP rule to filter pkts with MC DA MAC addr and route them to MGBE2_1 virtual n/w interface
./nvether_sample_app mgbe2_0 frp_add 2 01005E0000C8 1 0 0 0 0x2
```

Once the rules are set, start the application to send packets with MC MAC DA addr as 01:00:5E:00:00:C8 on Host1 and on Host2 observe the MC packets reaching it using capture tool and vice versa.

For more details on FRP rules, refer to FRP Validation under Networking.

Limitations

- > All interfaces used across the layer-2 bridge must have same MTU size configured. Packets transmitted by devices must be limited to the configured MTU size.

- > This solution is designed to support only low bandwidth use cases like Diagnostics System, etc. needed only in non mission mode.
- > The design assumption that both virtual interfaces added to the Layer-2 bridge are owned by a single VM.

3.8.16 TSN API

TSN is grouped into three basic key component categories that are required for complete real-time communication. There are multiple specifications that come under these 3 categories:

1. Time Synchronization i.e. 802.1AS(gPTP)
2. Scheduling and traffic shaping - i.e., 802.1Qav, 802.1Qbv, 802.1Qbu/802.3br, 802.1CB and 802.1Qci.
3. Selection of communication paths, reservation and fault tolerance - i.e., SRP (802.1Qat) and 802.1Qcc are key standards that define the system configuration of TSN networks.

Standards	Support on Orin SW + HW
802.1AS	YES
802.1 Qbu + 802.3br (egress path)	YES
802.1 Qbv (egress path)	YES
802.1 Qav (CBS) (egress path)	YES
IEEE 802.1Qat (SRP)	NO

This chapter provides sample application code for reference and documents the ioctl interface, which can be used to write customized applications to validate TSN specifications for Linux and QNX OS. Sample applications are used to configure end-to-end connection between 2 Ethernet interfaces.

Important Terms

Table 3.

Term	Description
TC	Traffic class value from 0 to 7
GCL	Gate control list (this is reliant to 802.1Qbv specification)
RQ	Frame Preemption Residue Queue. The Rx queue number to which the residual preemption frames must be forwarded. Preemption frames that are tagged and pass the SA/DA/VLAN filtering are routed and All other frames are treated as residual frames and are routed to the queue number mentioned in this field. The Queue-0 is used as a default queue for express frames, so this field cannot be programmed to a value 0.
APP	Application
OS	Operating system
DUT	Target device

3.8.16.1 IOCTL Interfaces

This chapter explains how to use driver provided private IOCTL for configuring hardware in Linux OS.

How to Use IOCTL Interfaces

```
/* include */
#include "ether_export.h"

static int open_socket(char *ifname)
{
    int sockfd;

    sockfd = socket(PF_INET, SOCK_STREAM, 0);
    return sockfd;
}

static void close_socket(int sockfd)
{
    close(sockfd);
}
```

```

int main(int argc, char *argv[])
{
    int sockfd;
    static char *if_name;
    struct ifreq ifr;
    struct ifr_data_struct data;
    struct osi_est_config cfg;

    sockfd = open_socket(argv[1]);
    if_name = strdup(argv[1]);

    /* Update all fields of osi_est_config from user input for example..*/
    cfg.en_dis = atoi(argv1);
        cfg.btr[0] = atoi(argv2);
        cfg.btr[1] = atoi(argv3);
        cfg.btr_offset[0] = atoi(argv4);
        cfg.btr_offset[1] = atoi(argv5);
        cfg.ctr[0] = strtol(argv6, NULL, 16);
        cfg.ctr[1] = strtol(argv7, NULL, 16);

    data.cmd = ETHER_EST_CFG;
    data.ptr = &cfg;
    strcpy(ifr.ifrn_ifrn_name, ifname);
    ifr.ifru_ifru_data = &data;

    ret = ioctl(sockfd, DWC_ETH_QOS_PRV_IOCTL, &ifr);
    if (ret < 0)
        printf("IOCTL Error in %s()\n", __func__);
    else
        printf("Configured successfully\n");
    close_socket(sockfd);
    Return ret;
}

```

Supported IOCTL Commands

```

/***
 * @addtogroup private IOCTL related info
 *
 * @brief MACRO are defined for driver supported
 * private IOCTLs. These IOCTLs can be called using
 * SIOCDEVPRIVATE custom ioctl command.
 * @{
 */
/** Line speed */
#define EQOS_GET_CONNECTED_SPEED      25
/** To set HW AVB configuration from user application */
#define ETHER_AVB_ALGORITHM          27
/** To get current configuration in HW */
#define ETHER_GET_AVB_ALGORITHM       46
/** To configure EST(802.1 bv) in HW */
#define ETHER_CONFIG_EST              49
/** For configure FPE (802.1 bu + 803.2 br) in HW */

```

```
#define ETHER_CONFIG_FPE          50
/** @} */
```

Data Structures

Data structure is part of the `drive-linux/include/` folder.

3.8.16.2 Sample Application



Note: This is sample development utility code and reference for development and must not be used in production.

802.1Qbu + 802.3br

IEEE 802.1 Qbu stops the transmission of long, non-critical frames to prioritize time-sensitive traffic, addressing the problem of transmission hogging. A major challenge for the timely transfer of critical messages is the presence of legacy traffic sharing the same network. Once a packet travels down a wire, it will block the wire from other packets until the end of the packet is reached.

To counter this issue, IEEE defined two standards- IEEE 802.1Qbu and IEEE802.3br, to support preemption. These standards, which build upon the TAS feature in 802.1Qbv, allow devices to preempt the transmission of non-TSN Ethernet frames (often legacy traffic) to prioritize high priority frames, while allowing the remainder of the interrupted frame to be sent later.

Syntax

```
nvethernet_sample <interface name> fpe <TC preemption mask> <RQ>
```

Parameters

- > TC preemption mask : preemption-bit corresponding to TC should have value 1 else express/default bit value 0.
- > RQ : Residual/default queue for unfiltered preemptable packets.This value can be any enabled Rx queue except rx queue 0.

Stats: ethtool -S <interface name>

- > mmc_tx_fpe_frag_cnt - Tx fragment count
- > mmc_tx_fpe_hold_req_cnt - Tx set-hold count
- > mmc_rx_packet_reass_err_cnt - Rx reassembly error
- > mmc_rx_packet_smd_err_cnt - Rx SMD error
- > mmc_rx_packet_asm_ok_cnt - Rx assembly ok

- > mmc_rx_fpe_fragment_cnt - Rx fragment count.

**Note:**

- > Rx Queue 0 can't be RQ
- > Please use nvethernet_sample <interface name> mmc_stats to get stats related to FPE
- > Use the primary interface i.e eqos_0, mgbe2_0, etc. for configuration
- > FPE and MACSEC don't coexist incase of MGBE HW due to HW limitation.
- > FPE get configured when peer device supports FPE and responds to verify packet(SMD-V).

For example:

- > To set preemption for TC 1 and TC 2 - (bit value representation rep 0x6), and RQ 2:
`nvether_sample_app <interface name> fpe 0x6 2`
- > To disable TX preemption:
`nvether_sample_app <interface name> fpe 0x0 2`

802.1Qbv

Traffic scheduling allows for different traffic classes to coexist with competing priorities on the same network. IEEE 802.1Qbv and IEEE 802.1Qbu work together to help manage this coexistence. IEEE 802.1 Qbv and IEEE 802.1Qbu can prioritize different traffic classes and enable time-sensitive data.

Devices/systems with IEEE 802.1Qbv can prioritize TSN Ethernet frames on a schedule, while non-TSN Ethernet frames to be transmitted on a best-effort basis around the TSN frames. The 802.1Qbv standard defines up to eight queues per port for forwarding traffic where each frame is assigned to a queue based on a QoS priority. To control the flow of queued traffic to a TSN enabled switch, this standard defines a time-aware shaper (TAS) mechanism that moderates queue traffic, preventing delays during scheduled transmission. Put simply, a gate in front of each queue opens at a specific point in time for time-sensitive traffic over standard (non-TSN) Ethernet packets. PTP should be initialized to make TSN working on DUT.

Syntax

```
nvether_sample_app <interface name> est <enable/disable> <base-time> <base-time offset>
<cycle time> <number of GCL entry> <GCL-entries>
```

Parameters

- > • enable - 1 and disable - 0
- > • base-time <nsec> <sec> put 0 0 if not sure - Value in decimal. If both values are 0 Driver will get PTP current time from MAC HW registers and add the base time offset into it.
- > • base-time offset <nsec> <sec> - Value decimal. This offset is given to start a new GCL from base time value + offset.
- > • cycle time <nsec> <sec> - Value in HEX. - ONE GCL full cycle time.
- > • number of GCL entry max index 255 - value in decimal

- GCL entries:
 - [0x<8 bit gate mask> <24 bit interval - value in ns>]
 - [0x<8 bit gate mask> <24 bit interval - value in ns>]
 - :
 - :
 - [0x<8 bit gate mask> <24 bit interval in ns>]

**Note:**

- > When EST and FPE are enabled, bit 0 of the gate mask represents set-and-hold. In this case preemption is enabled for TC 0.
- > When only EST is enabled, all bits of the gate mask represent open-and-close. In this case all TC are expressed TC.
- > Disable Tx flow control from device tree.
- > Use the primary interface i.e eqos_0, mgbe2_0, etc. for configuration.

Stats: ethtool -S <interface name>

- > const_gate_ctr_err - CGCE error count
- > head_of_line_blk_sch - Head of line block due to scheduling count
- > hlbs_q[0] - Head of line block due to scheduling count for TC 0
- > hlbs_q[1] - Head of line block due to scheduling count for TC 1
- > hlbs_q[2] - Head of line block due to scheduling count for TC 2
- > hlbs_q[3] - Head of line block due to scheduling count for TC 3
- > HLbs_q[4] - Head of line block due to scheduling count for TC 4
- > hlbs_q[5] - Head of line block due to scheduling count for TC 5
- > hlbs_q[6] - Head of line block due to scheduling count for TC 6
- > hlbs_q[7] - Head of line block due to scheduling count for TC 7
- > head_of_line_blk_frm - Head of line block due to size count
- > hlfq[0] - Head of line block due to size count for TCO
- > hlfq[1] - Head of line block due to size count for TC1
- > hlfq[2] - Head of line block due to size count for TC2
- > hlfq[3] - Head of line block due to size count for TC3
- > hlfq[4] - Head of line block due to size count for TC4
- > hlfq[5] - Head of line block due to size count for TC5
- > hlfq[6] - Head of line block due to size count for TC6
- > hlfq[7] - Head of line block due to size count for TC7
- > base_time_reg_err - Base time wrongly programmed in the past.
- > sw_own_list_complete - GCL switch happens successfully. This will increase this counter by 1.

For example:

- > Enable EST with cycle time 16,777,685 ns, 2 GCL entries where in 1st entry for TCO open for 470 ns and in 2nd entry no gate open for 16,777,685 ns.

```
nvether_sample_app eth1 est 1 0 0 0 10 0x10001d5 0 2 0x10001d6 0xffffffff
```

- > Example for FPE and EST enable with set-and-hold feature:

- Configure FPE as mentioned in 802.1Qbu + 802.3br
- Configure EST with set-and-hold feature like for case “Enable EST with cycle time 16,777,685 ns, 2 GCL entries where in 1st entry for TCO open for 470 ns with set-and-hold bit set and in 2nd entry no gate open for 16,777,215 ns”

```
nvether_sample_app eth1 est 1 0 0 0 10 0x10001d5 0 2 0x30001d6 0xffffffff
```

To check whether the new GCL is activated

As GCL configuration and GCL activation are 2 different things, please note programmed GCL gets activated based on Base time programmed. Please check results of ethtool -S <interface name> to check sw_own_list_complete value updated from last value or any error counter increased.

802.1Qav (CBS)

Originally developed to meet the bandwidth requirements of audio/video streams while preserving bandwidth for best effort traffic, IEEE 802.1Qav, known as Forwarding and Queueing for Time-Sensitive Streams (FQTSS) and later integrated into 802.1Q, defines a credit-based traffic shaper for reserving bandwidth for A/V streams in a bridged network.

In bridges, reserved A/V stream traffic is prioritized over best effort traffic as long as sufficient credits are available to transmit the higher-priority A/V data. As with end stations sourcing stream data, the credit-based traffic shaper results in stream data distributed evenly across a bridged network over time.

Syntax

```
nvether_sample_app <interface name> avb <qinx> <algorithm_value> <bw> <credit_control> <tcinx>
```

Parameters

- > qinx : Tx Queue index
- > algorithm value
 - 0 - default (strict priority for EQOS, ETS for MGBE)
 - 1 - cbs (credit based shaper)
- > bw : value in Mbits/sec of bw to be allocated
- > credit_control
 - 0 - disabled credit_control. The credit accumulates even when there is no packet waiting in TC<n> and another TC is transmitting.
 - 1 - enabled credit_control. When there is no packet waiting in TC<n> and other TC is transmitting, no credit is accumulated.

- tcinx : TC mapping for <qinx>

**Note:**

- > credit_control is applicable for cbs only. CBS configuration is per TC.
- > Use the primary interface i.e., eqos_0, mgbe2_0 etc. for configuration.

For example:

- > Give 20% bandwidth to TxQ/TC 2:

```
nvether_sample_app <interface name> avb 2 1 20 1 2
```
- > Reset to default tx selection algorithm for TxQ/TC 2:

```
nvether_sample_app <interface name> avb 2 0 20 1 2
```

VLAN (802.1Q)

In Linux, to create VLAN and MAP, the ingress and/or egress priority to SKB priority following Linux standard tools/utilities can be used. Example commands are:

```
ip link add link eth0 name eth0.5 type vlan id 5
ip addr add 10.0.1.6/24 brd 10.0.1.255 dev eth0.5
ip link set dev eth0.5 up

ip link add link eth0 name eth0.5 type vlan id 5 egress 0:1 1:2 4:4
```

In the last command above, the internal skb prio 0 is mapped to VLAN prio 1, skb prio 1 to VLAN prio 2 and 4 to 4.

You can also use the built in application vconfig:

```
vconfig set_egress_map [vlan-name] [skb_priority] [vlan_qos]
```

PTP (802.1AS) : 2 Step PTP

Nodes in the TSN network communicate with each other in real time and need a shared understanding of time to agree on corrective actions, recognize each other's state, and cooperate together.

The IEEE 802.1AS project created a profile of the IEEE 1588 PTP synchronization protocol for TSN. This profile will enable clock synchronization compatibility between different TSN devices. 802.1AS also addresses support for fault tolerance and multiple active synchronization masters.

Multiple profiles are supported.

Chapter 4. Flashing

The following sections describe how to flash and customize your system.

4.1 Flashing Basics



Note: You do not need to use sudo for the following commands:

- > bootburn.py
- > create_bsp_images.py
- > flash_bsp_images.py

However, if you choose to use sudo for one, use it consistently for the others. Do not switch between sudo/non-sudo usage.

This topic provides guidance on flashing preprocessed binaries.

4.1.1 Flashing AURIX from NVIDIA Orin

You can use the nv_aurix_check_fw script to check the AURIX firmware version on boot.

- > The script is located at
 - For QNX:
`<top>/proc/boot/nv_aurix_check_fw.sh`
 - For Linux:
`<top>/etc/systemd/scripts/nv_aurix_check_fw.sh`
- > If the AURIX firmware is the latest version, a corresponding message is displayed at the console.
- > If the AURIX Main firmware version requires updating, the AURIX update command runs automatically and updates the firmware to the latest version.

The correct hex file for flashing is chosen from the root filesystem at `/lib/firmware`.

- > After the Main AURIX firmware is updated, the version of the Update firmware is checked and the Update firmware is flashed if a newer version is found.

To check the firmware version again from the SoC:

- > Run the command:

- For QNX:
`/bin/ksh /proc/boot/nv_aurix_check_fw.sh`
- For Linux:
`sudo /bin/bash /etc/systemd/scripts/nv_aurix_check_fw.sh`

To update the AURIX firmware:

1. Ensure that both the Main (NV-IFW or AFW) AURIX firmware and the UPDATE(NV) firmware files are in the root filesystem.
2. Ensure that the files follow the naming convention as follows:

```
DRIVE-<branch>-<platform>-[NV/AFW]-Aurix-[UPDATE]-<Board>-
<major>.<minor>.<revision>[-SNAPSHOT-<snapshot>].hex
```

Where:

- > The path to the AFW firmware and UPDATE firmware is:
 - `/lib/firmware/*<platform>-AFW-Aurix-*`
 - `/lib/firmware/*<platform>-NV-Aurix-UPDATE*`
 - > NV indicates NVIDIA internal firmware.
 - > AFW indicates AUTOSAR firmware from the vendor Vector.
 - > <platform> is the board number, such as P3663.
 - > <FW_Type> is IFW / UPDATE / AFW.
 - > <Board> StepB.
 - > <major> contains one numeric digit.
 - > <minor> contains two numeric digits.
 - > <revision> contains two numeric digits.
 - > <snapshot> contains two numeric digits.
 - > Items in brackets [] are optional.
 - > Version numbers with 1.x.x indicate internal firmware; 5.x.x is the AFW from vendor Vector.
3. Update the firmware by running the command:

- > For QNX:
`</bin/ksh/ /proc/boot/nv_aurix_check_fw.sh -auto_update`
- > For Linux:
`<sudo /bin/bash /etc/systemd/scripts/nv_aurix_check_fw.sh -auto_update`

Where the path to the AFW and UPDATE firmware is as follows:

- > `/lib/firmware/*<platform>-AFW-Aurix-*`
- > `/lib/firmware/*<platform>-NV-Aurix-*`

Updating the firmware includes:

- > Comparing the firmware version between the flashed version and the one on the root filesystem.

- > Updating the firmware if a higher firmware version is in the root filesystem.
 - > Resetting the platform.
4. When the platform boots again, repeat the command to reset it a second time.
- After two resets, the AURIX MAIN firmware is updated.
5. Rerun the command to update the UPDATE firmware.

**Note:**

The AURIX flashing program/erase cycle is 1000. To avoid wear on the AURIX flash components, flash the firmware only when necessary.

The flashing of AURIX MAIN and UPDATE firmware requires a maximum of two SoC resets, and three invocations of the `nv_aurix_update` application.

6. Board supports only the Vector AFW (Versions 5.x.x) and NV internal firmware (version 1.x.x). When the Vector AFW hex file is not packaged in the `/lib/firmware` folder, the update script automatically chooses NV internal FW as an alternative.

4.1.2 Bind Steps for NVIDIA Orin System on a Chip (SOC)

For information about how to bind an NVIDIA Orin system on a chip, see See the "AV PCT Configuration" topic in the *NVIDIA DRIVE OS Linux PDK Developer Guide*.

4.1.3 Flash Steps for NVIDIA Orin System on Chip (SoC)

Use the bootburn utility to flash platform boards with the boot loader, kernel, and file system. The bootburn utility is provided as part of the Foundation package.

Bootburn

Bootburn is a Python script that:

- > Boots the device in the recovery-mode with RAMDisk
- > Signs or generates the images on the host
- > Transfers the images to target using ADB
- > Flashes the boards

Bootburn automatically resizes the rootfs image on the target to optimize the available storage on the partition dedicated to it, either on eMMC or NOR.

Bootburn supports a host NFS share as the mount point for the target rootfs. This feature provides a convenient way for developers to copy files between the target and host for rapid development. In this case, the uncompressed target rootfs directory on the host is leveraged directly by the target.

Bootburn relies on the Foundation services to provide the flashing and boot loader functionality. You configure it with the CFG file rather than with bootburn command-line parameters; the bootburn utility supports limited command-line options.

Bootburn Command Usage

Run the `bootburn.py` script from a shell on your host Linux system by executing the command:

```
./bootburn.py <options>
```

Bootburn Options

Options	Description
-b <p3710-10-a01 p3710-10-a03 p3710-10-a04 p3710-10-s05 p3710-12-a01 p3710-12-a03 p3710-12-a04 p3710-12-s05 p3663- a01 p3663-01-a02 p3663-02-a02 p3898- a01> (lowercase b)	<p>Provides the board name and revision. Possible board names include:</p> <ul style="list-style-type: none"> > p3710-10-a01 > p3710-10-a03 > p3710-10-a04 > p3710-10-s05 > p3710-12-a01 > p3710-12-a03 > p3710-12-a04 > p3710-12-s05 > p3663-a01 > p3663-01-a02 > p3663-02-a02 > p3898-a01 <p>Not all boards are supported in Safety and Standard. For more information, see the API Reference.</p>
--board_config <board_name>.json	Can be used with <code>bootburn.py</code> , <code>create_bsp_images.py</code> , and <code>flash_bsp_images.py</code> . <code>board_name</code> must include absolute path.
-d <partition_name> <dtb_file> (lowercase d)	<p>Specifies the DTB file to flash and the target partition. The default file is defined in <code>BoardSetFilePathsAndDefaultValues</code>.</p> <p><code>bmp-fw-dtb</code> and <code>kernel-dtb</code> may be specified simultaneously.</p> <p>Note: This option is not intended to specify <code>kernel-dtb</code> in Hypervisor.</p>

Options	Description
-m (lowercase)	Flashing Modular Diagnostic Software (MODS). This option enables mods-specific overrides.
-o (lowercase o)	<p>Skips flashing of recovery partitions. Used by minicom.</p> <p>WARNING: If you are using minicom with -o option, minicom does not initialize the port and therefore does NOT check or acquire the lock files for the communication port. If minicom is running in the background and you run bootburn with the -x option, bootburn acquires the lock BEFORE accessing the communication port. So, when bootburn attempts to communicate with AURIX it fails.</p>
-p <key_file_path> (lowercase p)	Signs the boot loader, secure OS (TOS), kernel and BCT binary, then flashes the device. <key_file_path> must be the full pathname of the key_file.
--hsm <key_string>	<p>Can be used with bootburn.py and create_bsp_images.py. Tells Bootburn what keys will be used in HSM mode. Key String is Key + Encrypt Keys. Key is rsa. Encrypt Keys are sbk. Example:</p> <p>--hsm rsa</p>
-q (lowercase q)	Flashes QNX images.
-s (lowercase s)	<p>Skips flashing the file system.</p> <p>Use the -s flag only when the target file system is already flashed.</p>
-t <output_directory> (lowercase t)	Specifies the output directory where images to be flashed onto the target are to be stored. The default directory is named _temp_dump, and is present in the flashing directory.

Options	Description
-x	<p>Specifies the communication port for AURIX to put the SoC in recovery mode automatically.</p> <p>With this -x option, bootburn performs target validation for the user provided board name on the command line, with the -b option, before flashing the board.</p> <p>Bootburn generates a unique ID from the target Inforom data. The allowed target ID list for the user provided board is generated from BOM data file included in the package. Bootburn aborts flashing if the target unique ID is not found in the list of generated unique IDs for the provided board name. This feature is supported on P3710, P3663, and P3898 boards with Inforom system object version 3 or above.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Note: On boards with inforom, the bct sku info is updated with the information in the inforom. </div> <p>If the -x option is used during the AURIX setup, at the end during the AURIX reset of the flashing procedure there may be output messages of the form: "Killed sudo cat \$1_Aurix > \$p_FlashFiles/\$1_AurixLogFile". These messages are not errors and can be safely ignored.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Warning: If minicom is used with the -o option, minicom does not initialize the port and therefore does NOT check or acquire the lock files for the communication port. <ul style="list-style-type: none"> > If minicom is running in the background and you run bootburn with the -x option, bootburn acquires the lock BEFORE accessing the communication port. So when bootburn attempts to communicate with AURIX it fails. > If you quit minicom forcefully it may not fully stop the process and it will be lingering in the background. > When bootburn attempts to connect in this state it will acquire the port but the output will still be directed to the minicom running in the background. > Avoid using the -o option when using minicom. </div>

Options	Description
-B <boot_device> (uppercase B)	Specifies the boot device, which must be either qspi or emmc. qspi is the default.
-C (uppercase C)	Specifies use of debug binaries for the boot loader.
-D (uppercase D)	Directs bootburn debug output to stdout along with the tool's regular output. If -D is not specified, debug output goes to a temporary file.
-E (uppercase E)	Enables DRAM ECC.
-I <bus_id> <device_id> (uppercase I)	<p>Flashes a specific device when multiple devices are in recovery.</p> <p>To get the bus and device ID of each device in recovery, enter the lsusb command on the host. For example, if lsusb gives the following output:</p> <pre>Bus 003 Device 105: ID 0955:7018 NVidia Corp. Bus 003 Device 104: ID 0955:7018 NVidia Corp.</pre> <p>Then flash the second device with the -I option as in this example:</p> <pre>-I 003 104</pre>
-L (uppercase L)	Enables low-power modes. This option requires that the spe-fw and WBO firmware packages be included in the bootburn configuration because low-power modes require them.
-M (uppercase M)	Specifies development version firmware.

Options	Description
-R (uppercase R)	<p>Specifies RCM boot support, where the device boots without being flashed. When the target is in recovery mode, this option causes the flashing script to download all binaries, other than BCT, from the host. The target then reads the BCT from storage and other binaries from RAM. It boots the kernel directly on RAM.</p> <p>This option is much faster than when flashing also occurs. It is especially useful during debugging.</p> <p>Use the -R flag to initiate RCMBOOT with any of the commands mentioned in this table.</p>
-T (uppercase T)	<p>Enables tracing. Tracing logs are stored under the bootburn directory as log_trace*.txt.</p>
--safety	<p>Execute bootburn scripts in safety mode. Only a limited set of options is available for use in safety mode. For more information, see the Safety Use Cases for Bootburn Scripts chapter in the <i>NVIDIA DRIVE OS 6.0 Safety Developer Guide</i> for more information.</p>
-U	<p>Pass in the UFS provision configuration file.</p>
-V (uppercase V)	<p>Specifies read-back verification, where the binary images are read back (after writing) from the target's storage and compared with the original binary images. The two sets of images are compared to detect discrepancies.</p>
--customer-data	<p>Specified customer data such as skuinfo and others to be updated during flashing</p> <p>See <i>flashing_customer_data.docx</i> for information on how to use it.</p>

Passing Additional Parameters

To pass additional kernel parameters, modify the `os_args` parameter of the `kernel-dtb` partitions in:

```
drive-foundation/
virtualization/pct/<pct>/qnx/qnx_<guest>/
qnx_vm1_storage_emmc.cfg
```

4.1.4 Flashing with Docker

The NVIDIA DRIVE® OS Docker build image contains a python script called `flash.py` to flash the target.

Usage

```
flash.py [-h] [--disable-full] [--pct-variant PCT] [--tegra TEGRA] [--clean]
[--recovery-timeout [TIMEOUT]] [--bind-opt BIND_OPT] [--opt-arg OPT_ARG] AURIX_PORT
BOARD
```

The following table shows required and optional arguments:

Argument	Description
BOARD	Specifies the target board base name.
AURIX_PORT	Specifies AURIX Port for the desired board.

Options

Use the following `flash.py` options to override the default parameters:

Option	Description
--tegra TEGRA	Specify the NVIDIA Tegra® chip to flash. The default is Tegra A.
-h, --help	Shows usage/help output and exits.
--disable-full	Flash single bootchain. Otherwise, will flash both bootchains.
--pct-variant PCT	Specify the PCT variant to flash.
--clean	Perform clean flash to re-initialize persistent partitions.
--recovery-timeout [TIMEOUT]	Specify the time required for the device to be available after it is set to recovery.
--bind-opt BIND_OPT, -b BIND_OPT	Specify arguments to pass through to bind partitions utility.
--opt-arg OPT_ARG, -o OPT_ARG	Specify argument to pass through to bootburn flashing utility.

Examples

- Flash the default configuration for p3710 the Docker image on board connected to /dev/ttyACM1:

```
# ./flash.py /dev/ttyACM1 p3710
```

- > If a PCT is required to be specified (for QNX, QNX Safety, Prod Linux, and so on), pass the PCT variant as an argument:

```
# ./flash.py --pct-variant prod /dev/ttyACM1 p3710
```

- > Additional Make arguments can also be passed through for the binding phase. For instance, to set R/W permissions on Root FS for the NDAS use case.

```
# ./flash.py --bind-opt '-u ndas' --bind-opt 'OS_ARGS_MOUNT_PER=rw' /dev/ttyACM0  
p3710
```

or

```
# ./flash.py -b '-u ndas' -b 'OS_ARGS_ROOT_MOUNT_PER=rw' /dev/ttyACM0 p3710
```

- > Additional arguments can be passed through for the flashing phase. For instance to disable UART on target for rcm-boot and cold-boot:

```
# ./flash.py --opt-arg '--disable_uart' /dev/ttyACM0 p3710
```

or

```
# ./flash.py -o '--disable_uart' /dev/ttyACM0 p3710
```

4.1.5 Flashing AURIX

AURIX is intended for debugging and sanity checking purposes.



Note: In this section, <top> refers to the root directory of the guest VM.

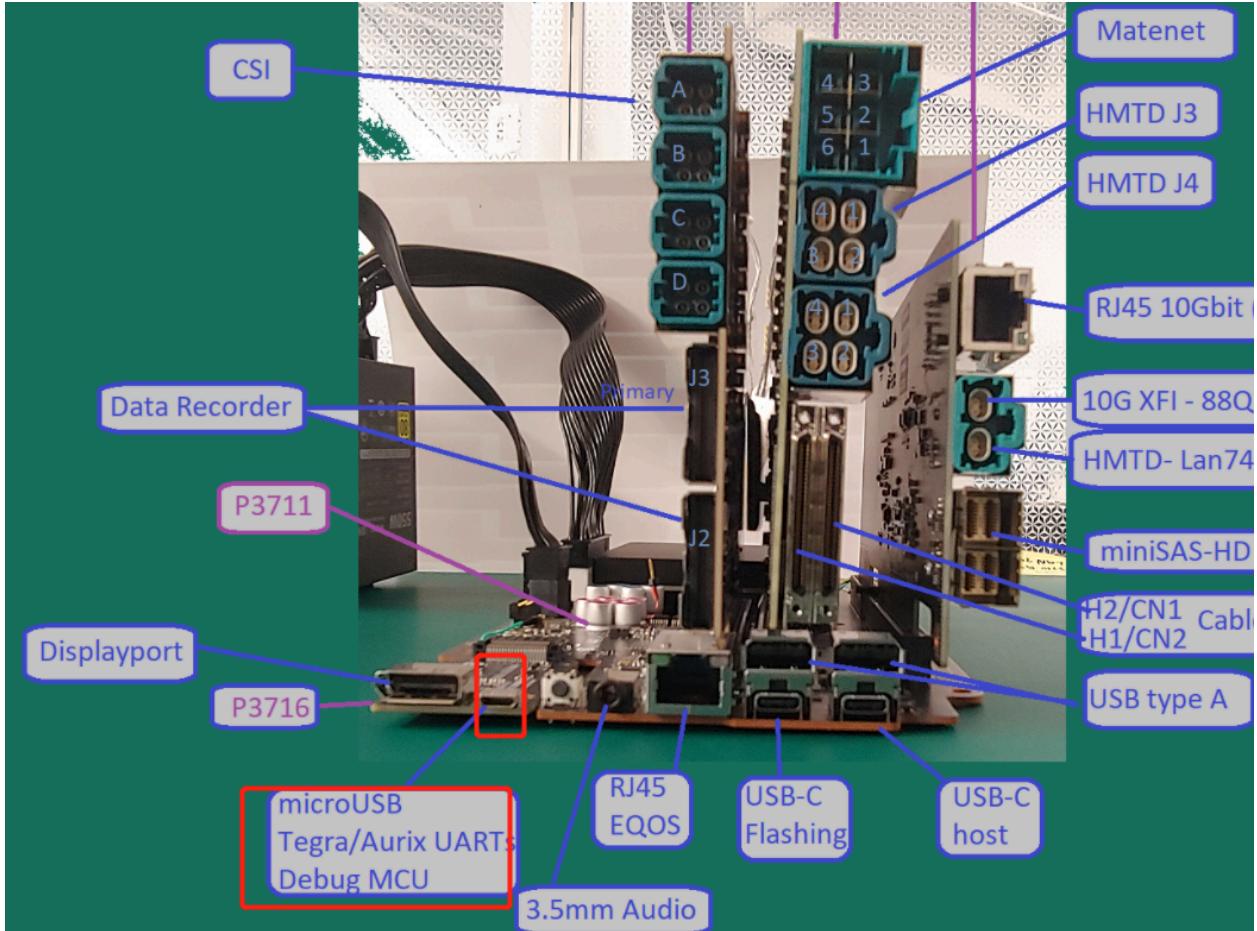
4.1.5.1 Flashing AURIX from Windows Host with Infineon Memtool

Step 1: Target Setup

On P3710

P3710 uses TOPO MCU to provide Aurix UART console access; you should connect microUSB to your windows host machine as shown in the following figures.

You should see a list of /dev/ttyACM*. Usually Aurix is the 2nd in the list of /dev/ttyACM*, which is /dev/ttyACM1 if starting from /dev/ttyACM0. You should find the Aurix console first before flashing.



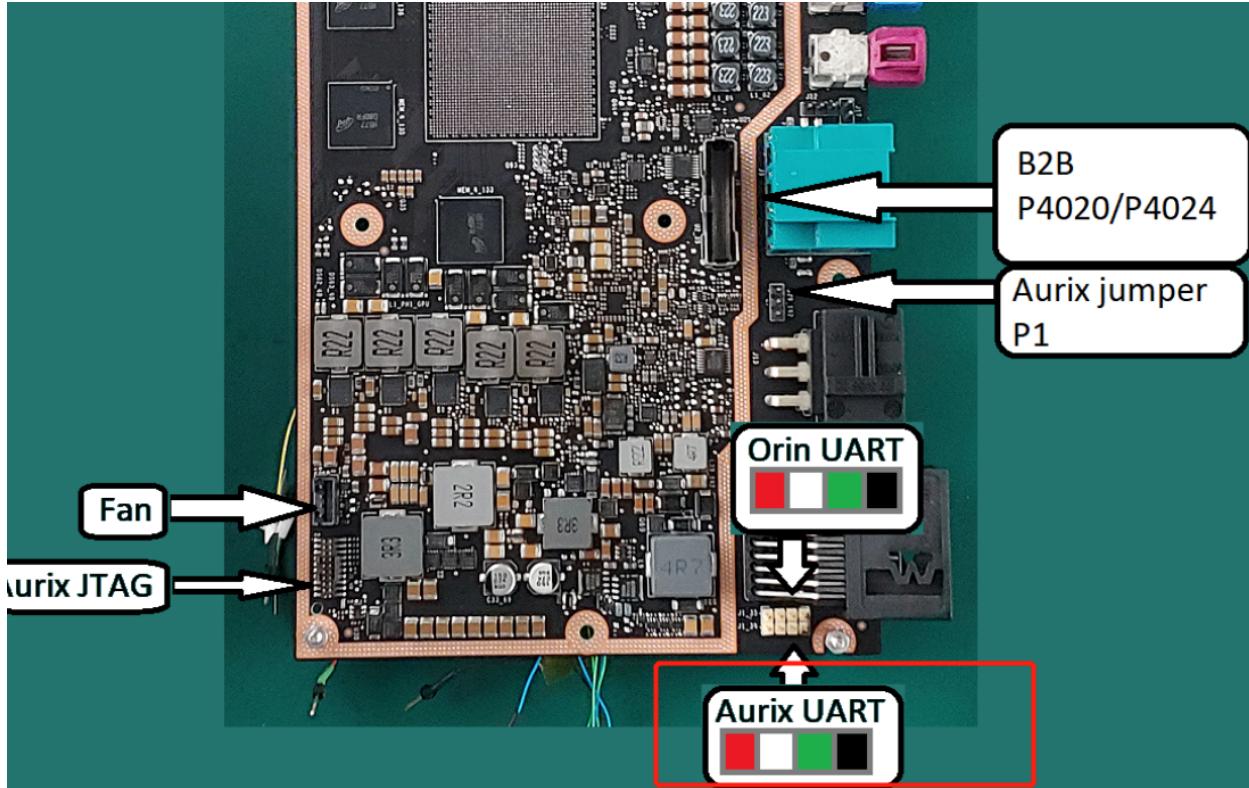
On P3663

P3663 does not have TOPO MCU for Aurix UART console; instead, the Aurix UART signal (VCC/TX/RX/GND) is routed to the jumper below.

You should connect the Aurix UART console jumper to

- > USB-UART cable
- > Windows host

In Windows host, you should see `/dev/ttyUSB*` for Aurix console.



On P3898

P3898 does not have TOPO MCU for Aurix UART console; instead, the Aurix UART signals (VCC/RX/TX/GND) are routed to the jumper J1_34 as shown below.

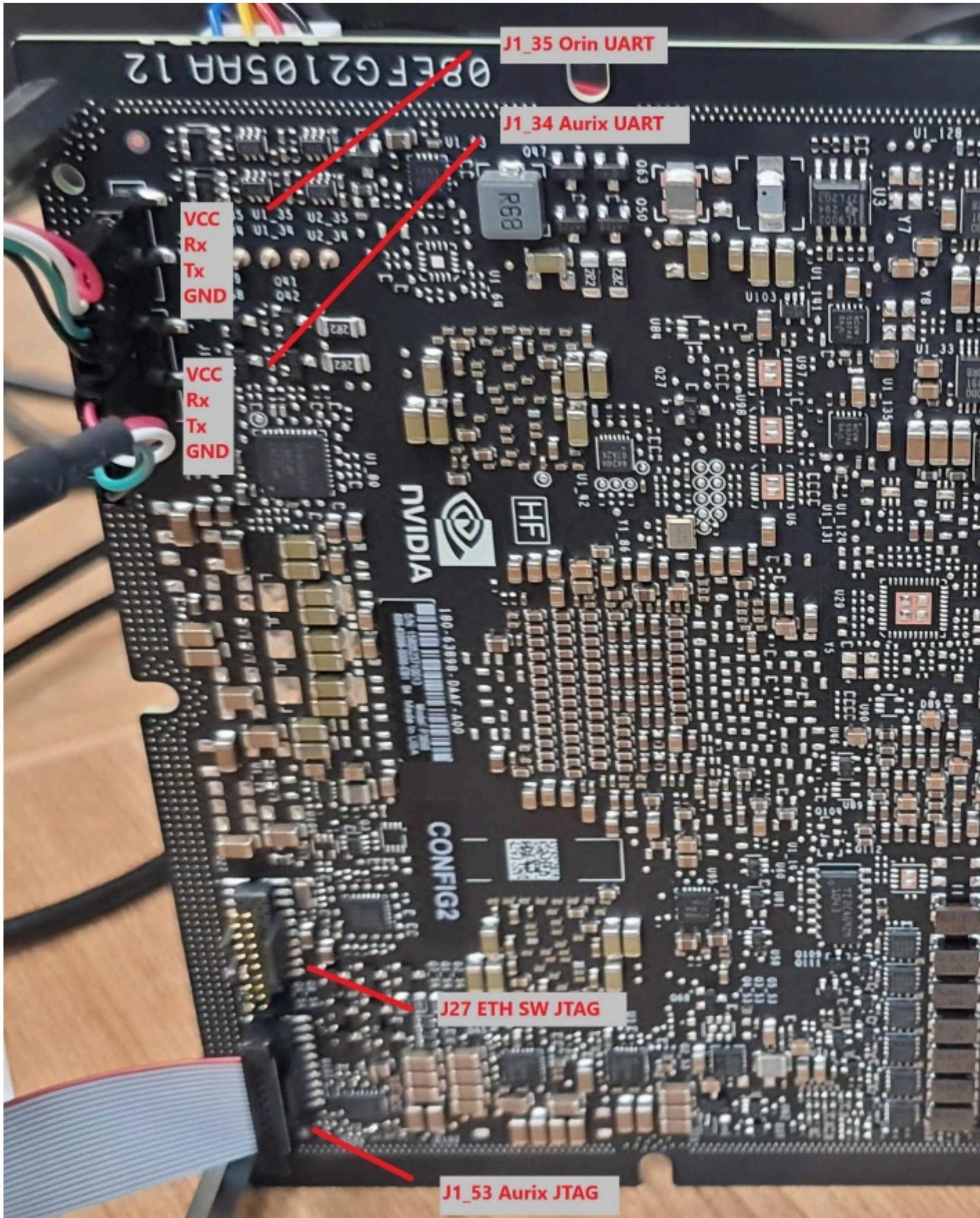
You should connect the Aurix UART console jumper to:

- > USB-UART cable
- > Windows host



Note: The Rx and Tx pins indicated below are respectively input and output from AURIX, so connect the USB-UART cable pins accordingly. For example, connect USB-UART Tx to Rx on board.

On the Windows host, you see `/dev/ttyUSB*` for AURIX console.



Step 2: Host Setup

Use a Windows machine for this upgrade.

1. Download and install the latest memtool from [Infineon](#).
2. For P3663 and P3710, use the UART cfg file below:

```
[Main]
Signature=UDE_TARGINFO_2.0
Description=TriBoard with TC39x B-Step (BSL/ASC)
MCUs=Controller0
Architecture=TriCore Aurix
Vendor=Starter Kits (Bootstrap Loader)
Board=

[Controller0]
Family=TriCore
Type=TC39xB
Enabled=1
IntClock=100000
ExtClock=20000

[Controller0.Core0]
Protocol=MMTC
Enabled=1

[Controller0.PCP]
Master=Core
Enabled=0

[Controller0.Core0.MmTcTargIntf]
MonType=ASC
BaudRate=57600
PortType=COMX
PortSel=COM4
ExecInitCmds=0
ReqReset=0
ReqResetMsg=
ResetOnConnect=1
ResetWaitTime=500
ExtStartMode=0
KLineProt=0
UseRS232Drv=1
CanPortNum=1
AssureSendOfComPort=0
Stm32AscBaudrateForConnect=0
CheckAckCode=1
AlwaysEINIT=0
UseExtMon=0
MonitorPath=
UseExtMon2=0
Mon2Path=
Mon2Start=0xFFFFFFFF
SCRMSupport=0
```

```
SCRMBaudRate=0
RSTCON_H=0x0
S0BRL=-1
UseChangedBaudRate=0
Sv2PLLCON=0x7103
Sv2ASC0BG=0xFFFF
Sv2CANBTR=0xFFFF
TcP11Value=0x0
TcP11Value2=0x0
TcP11Value3=0x0
TcAscBgValue=0x0
TcCanBtrValue=0x0
XC2000ScrmClock=40000000
MaxReadBlockSize=0
BootPasswd0=0xFEEDFACE
BootPasswd1=0xCAFEBEEF
AurixEdBootWorkaround=0

[Controller0.PFLASH0]
Enabled=1
EnableMemtoolByDefault=1

[Controller0.PFLASH1]
Enabled=1
EnableMemtoolByDefault=1

[Controller0.PFLASH2]
Enabled=1
EnableMemtoolByDefault=1

[Controller0.PFLASH3]
Enabled=1
EnableMemtoolByDefault=1

[Controller0.PFLASH4]
Enabled=1
EnableMemtoolByDefault=1

[Controller0.PFLASH5]
Enabled=1
EnableMemtoolByDefault=1

[Controller0.DF_EEPROM]
Enabled=1
EnableMemtoolByDefault=1

[Controller0.DF1]
Enabled=1
EnableMemtoolByDefault=0

[Controller0.DF_UCBS]
Enabled=1
EnableMemtoolByDefault=0
```

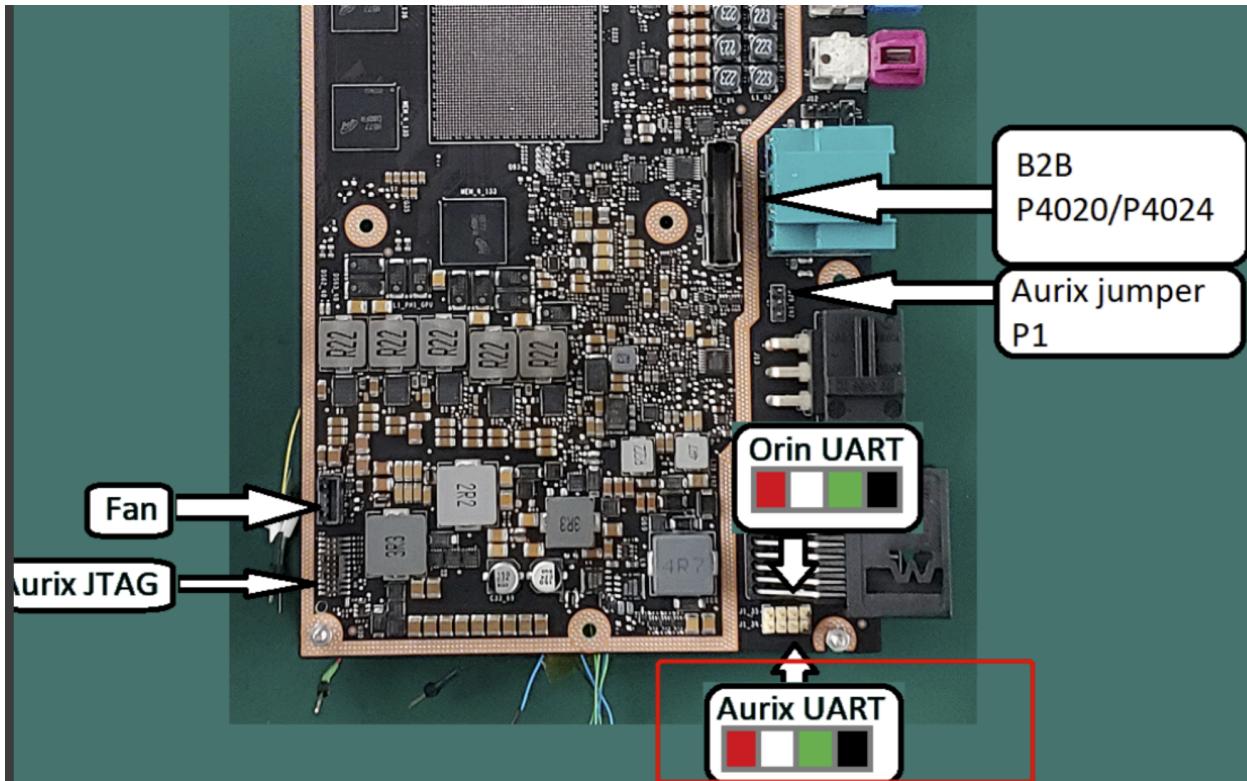
```
[Controller0.Core0.MmTcTargIntf.InitScript]
[FlashMod_DF_UCBS]
Enabled=1
```

If you use miniwiggler to upgrade (a JTAG debugger), you should use the miniwiggler one.



Note: This is the same configuration file as the Xavier platform step B AURIX. For UART, the TriBoard_TC39xB_ASC_BSL.cfg configuration file is located in the delivery folder.

- For P3898, the Miniwiggler configuration file to be used for flashing through UART is located in the delivery folder and is named TriBoard_TC36x.cfg.



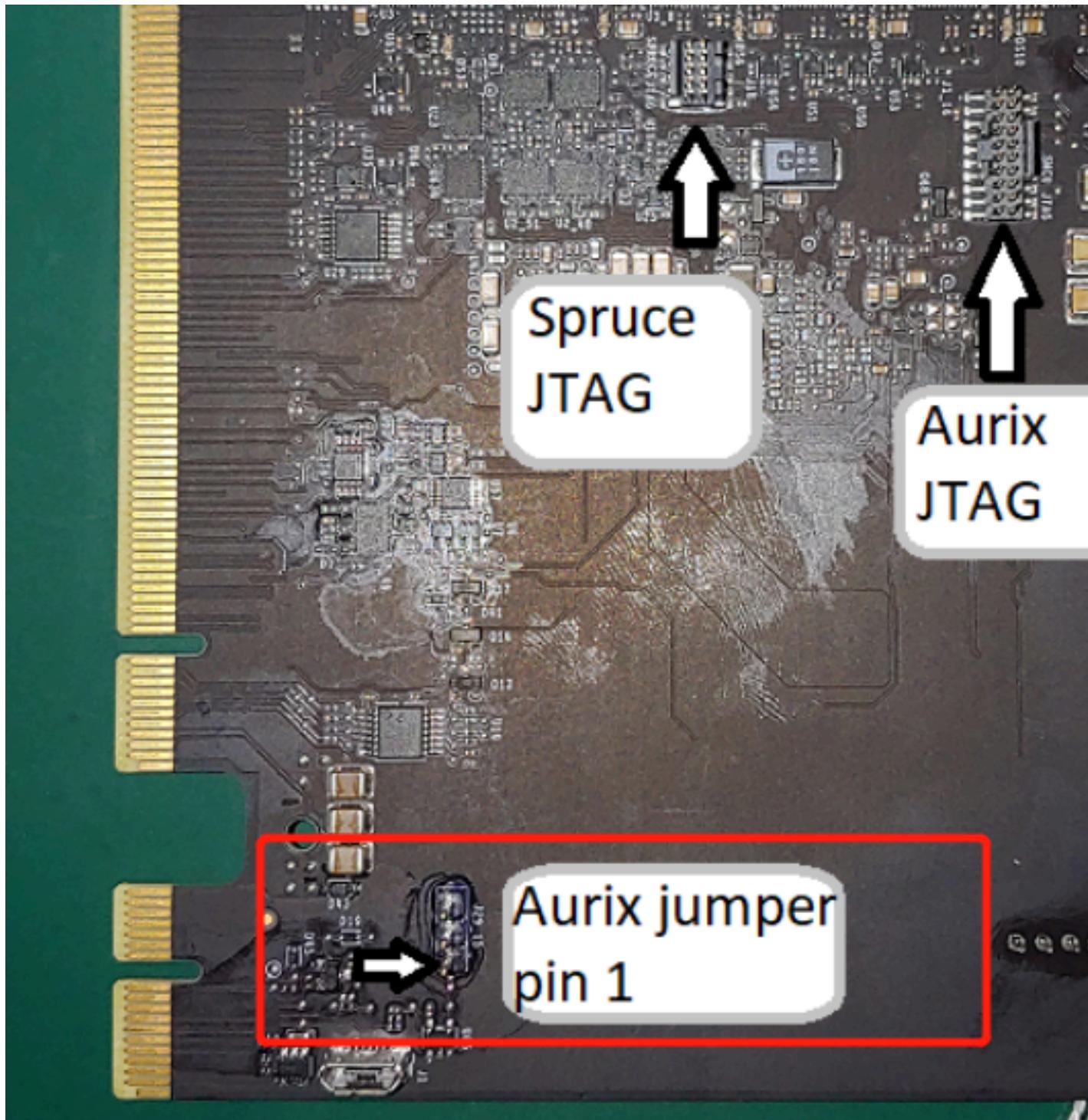
P3710:

- Find the jumper below and insert JP 1-2 (short pin 1 and pin 2).
- Power cycle the P3710 system.

This can put Aurix into recovery mode. In this mode, Aurix console does not show any message.



Note: Close any tool that opening Aurix console before proceeding to the next step.



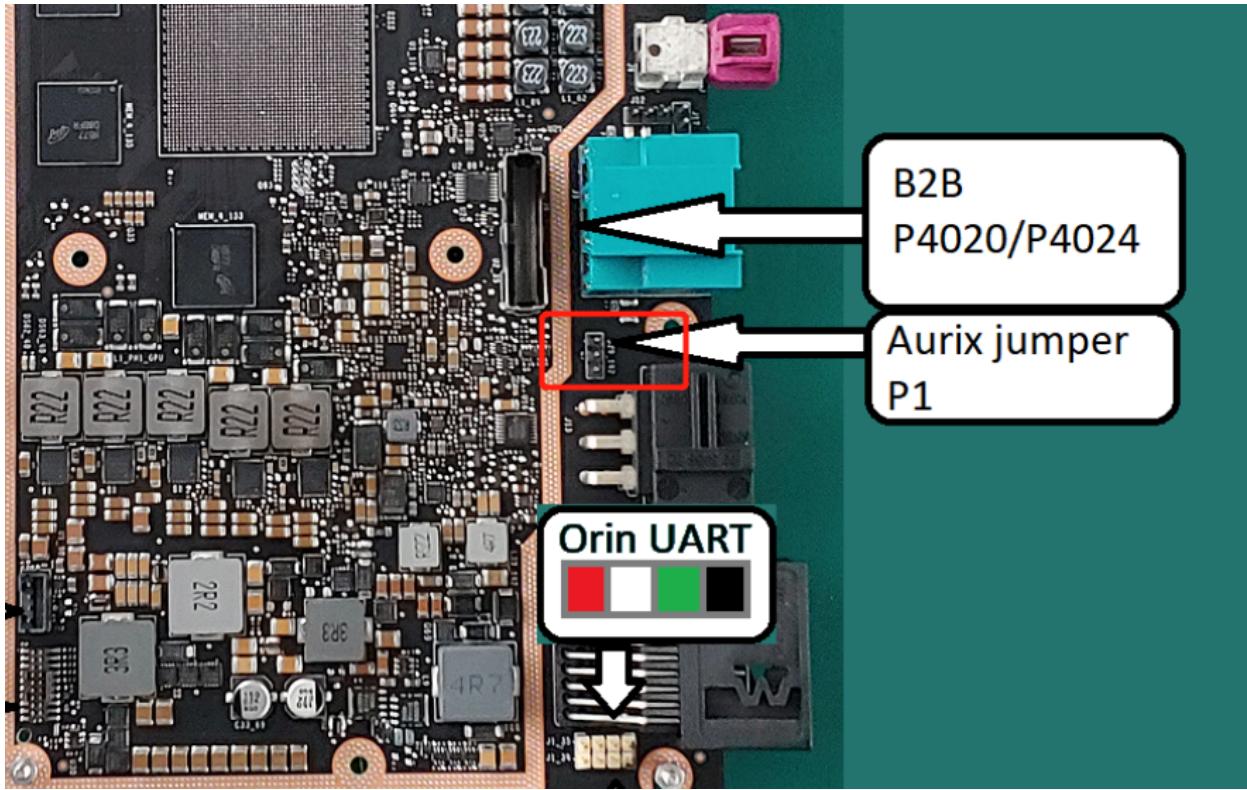
Step 3: Put Aurix into Recovery Mode

First put Aurix into recovery mode.

P3663:

1. P3663 does not have TOPO, so you need to insert pin 1 and pin 2 for the jumper.

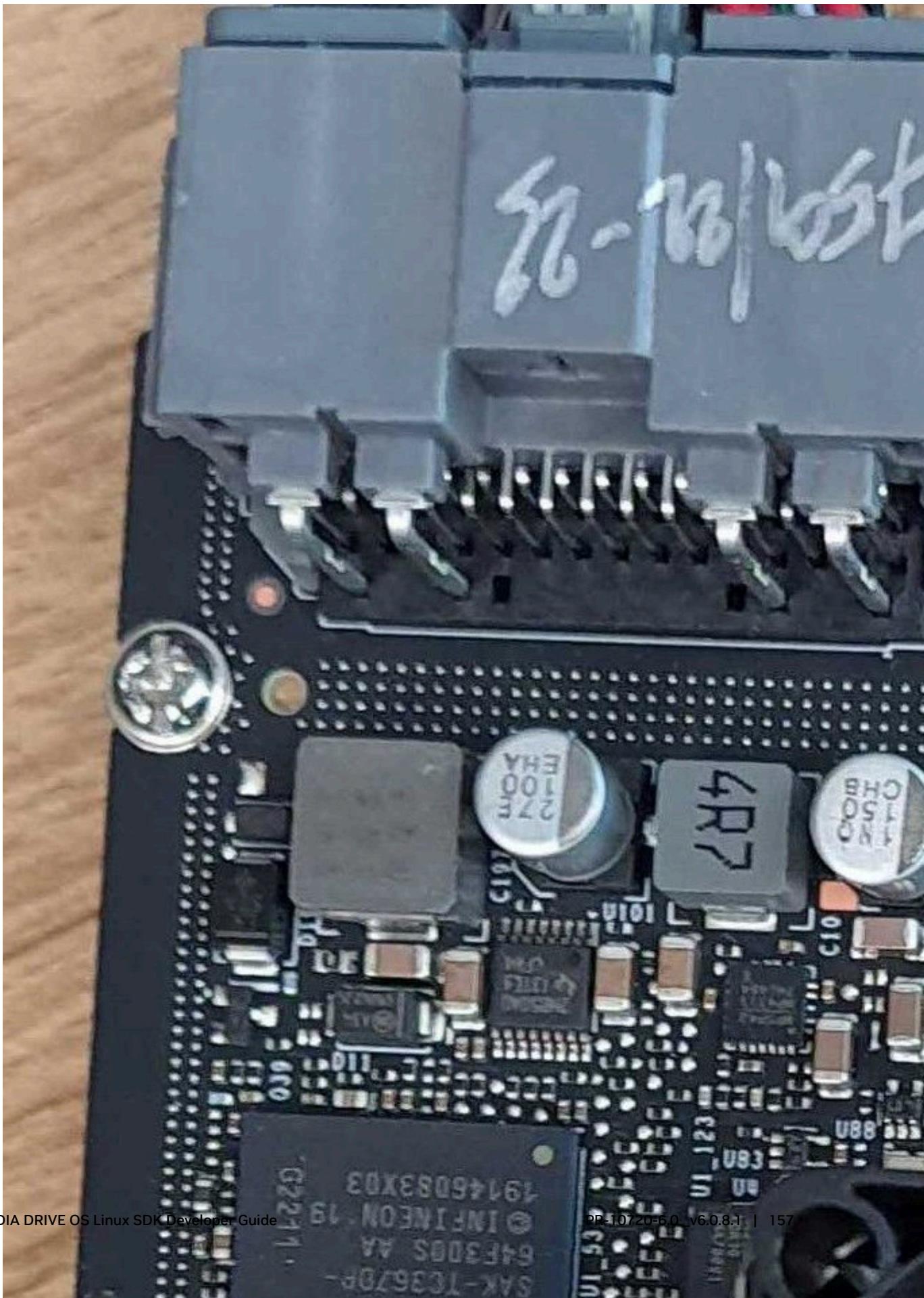
2. Power cycle P3663.



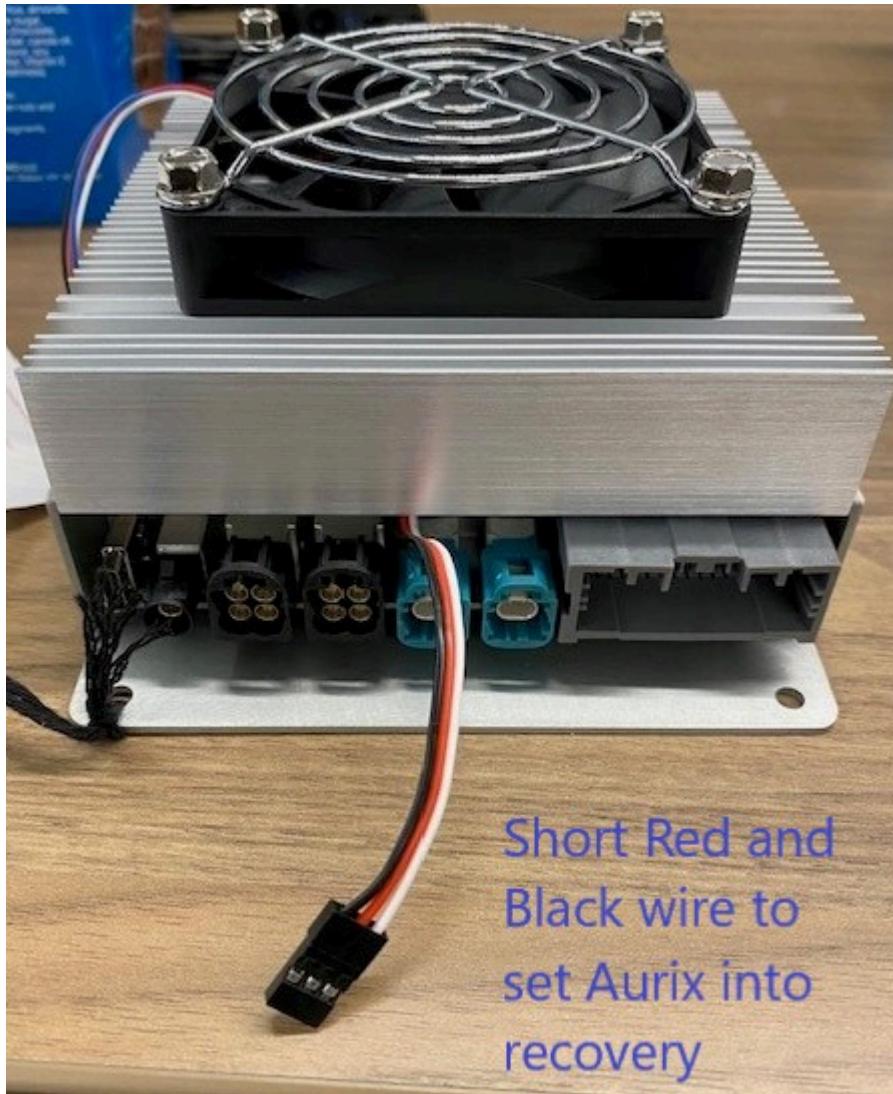
P3898:

1. Short J1_57 pins 1 and 2 to set HWCFG3 = 0 for UART boot. Please refer the figure below for location of the jumper and pins to be shorted. As shown in the figures below, access to these pins vary across P3898 board revisions.
2. Power cycle P3898.

P3898 (A00)



P3898 (B00)

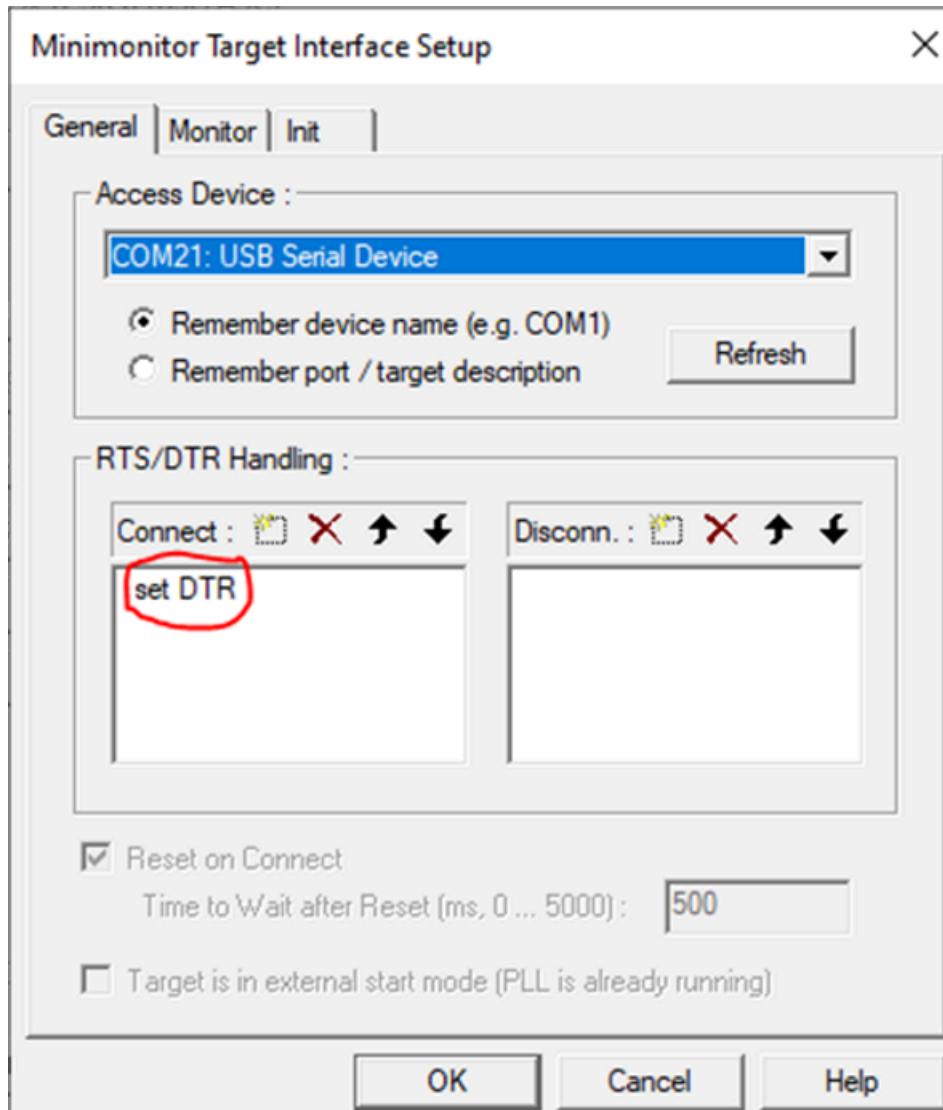


Step 4: Connect Memtool to Aurix

To start the memtool,

1. Choose **Target > Change** and select the folder where you have saved the configuration file in step 2.
2. Select **Triboard with TC39x (BSL/ASC)** and click OK. For P3898, select **Triboard with TC36x (BSL/ASC)**.
3. Choose **Target > Setup** and select **Remember device name (e.g. COM1)**.
4. In Access Device, choose the COM port as identified in step 1.
5. Add “set DTR” to the RTS/DTR Connect settings as shown below.

If you have tried to connect but failed. Restart memtool and P3710, and ensure you have this setting.



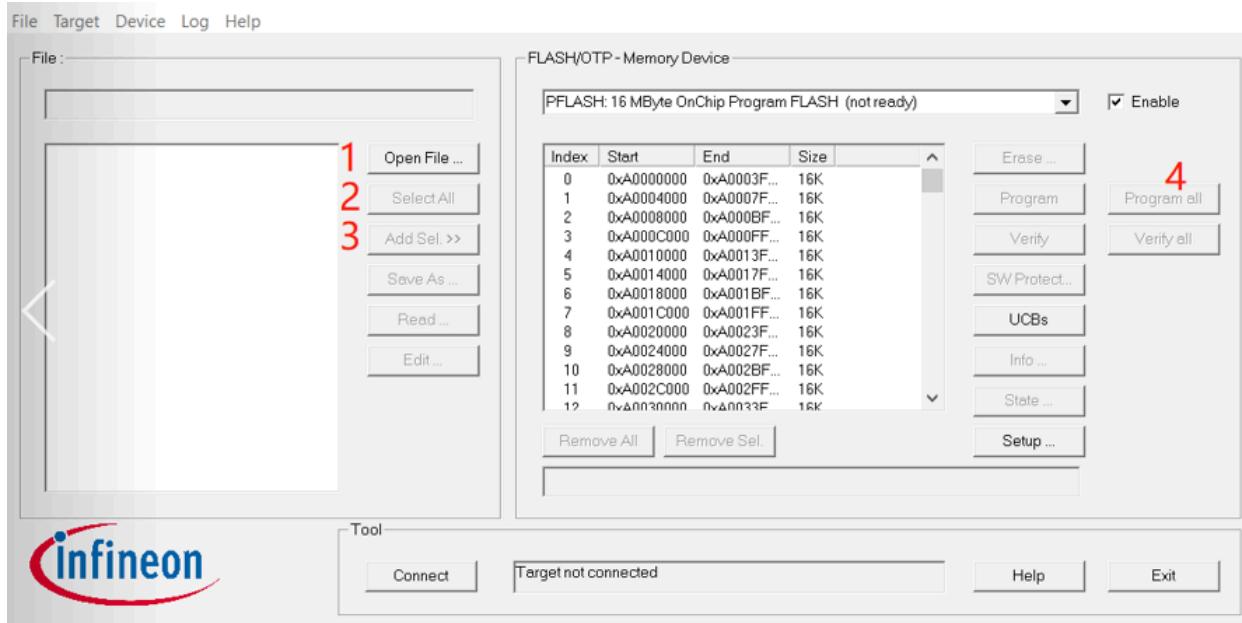
Step 5: Start Flashing

1. Click Open File and select the file.
2. Click Select All.
3. Click Add Sel. >> to add all the file content to the flash area.
4. Click Program all to start the flashing.

The flash should start and finish.



Note: Do not use the Erase button in the memtool.



Note: For P3898 with TC36x AURIX variant, the PFLASH size is shown as 4 MByte.

4.1.5.2 Flashing AFW from Orin Using Force Update

AFW can also be flashed using the update script from the NVIDIA Orin console.

Prerequisites: The following firmware versions on MCU are required:

- > Internal firmware (IFW) version 1.46.4 or the latest
- > Update firmware (UFW) version 1.46.4 or the latest
- > For P3898, IFW and UFW version 1.60.2 or later

Command:

- > For Linux:

```
sudo /bin/bash /etc/systemd/scripts/nv_aurix_check_fw.sh -force_update
AFW_Firmware_to_be_Programmed
```

- > For QNX:

```
/bin/ksh /proc/boot/nv_aurix_check_fw.sh -force_update AFW_Firmware_to_be_Programmed
```

Example:

```
sudo /bin/bash /etc/systemd/scripts/nv_aurix_check_fw.sh -force_update DRIVE-V6.0.0-
P3663-AFW-Aurix-StepB-5.00.01.hex
```

Run this command twice:

1. On first execution of this command, control switches from IFW to UFW leading to NVIDIA Orin reset during switching.
2. After NVIDIA Orin starts up completely, repeat this command, which starts AFW flashing.

4.2 Flashing Customization

This section provides guidance on customizing flashing.

- > Bootburn
- > Kernel
- > Prebuilt binaries
- > Offline binaries

Select Boot Chain Platform Configuration

Within the board support package (BSP) with Virtualization support the top-level configuration file `global_storage.cfg` defines the sub configuration file used to create the target BSP. Currently, this defaults to `boot_chain_storage.cfg`. As an example, if you run the `bind` command:

Usage

```
./make/bind_partitions [options] [pct name] [make target]
```

Required Options

```
BOARD=p3710-10-s05
OS=qnx
```

Options

List of all options:

```
./make/bind_partitions [options] [pct name] [make target]
```

This wrapper script calls `boot_blob.mk`.

Short Code	Long Name	Description
-b	--board	Specifies the board variant.
-c	--chip_variant	Specifies the chip variant.
-d	--gos_dtb	Builds GOS device tree blobs.
-e	--cpu_type	Type of CPU to spoof.
-f	--tos_flavor	TOS type (tzvault or other variant).
-g	--bpmp_dtb	Builds BPMP device tree blobs.
-h	--help	Shows this help message.
-i	--ssv_test	Virtualization test variant.

Short Code	Long Name	Description
-n	--nfs_boot	Boots up Linux Guest OS mounting rootfs through NFS with specify Ethernet interface.
-p	--pct_variant	Select specify PCT variant profile file in DRIVE_AV/DRIVE_IX profile.
-r	--storage_size	Type of storage config to use (32gb, 64gb, no_ufs).
-s	--soc_id	Specifies SOC ID in GOS device tree and rebuilds GOS device tree.
-t	--tos_target	Specifies the TOS target.
-u	--ndas_usecase	Specifies the NDAS enable or EcoSystem/General usecases.
-v	--verbose	Verbose output.
-w	--power_profile	Specifies the power profile.
-x	--chain_a	Specifies chain-A config file.
-y	--chain_b	Specifies chain-B config file.
-z	--fskp_fuse_burning	Enables fskp partition in Linux configuration for fuse burning.
N/A	--oot_kernel	Builds DTB for OOT kernel.
N/A	--scl_arg	Specifies argument for SCL components.

[pct name]: Specifies the PCT name to use.

[make target]: make target.

Examples

```
if [[ "$QNX_PDK" == "qnx-safety" ]]; then
echo " BINDING SAFETY ... "
sleep 5
cd $LOCAL_PDK/drive-foundation-safety
./make/bind_partitions -b $BOARD -p prod_debug_extra qnx clean
./make/bind_partitions -b $BOARD -p prod_debug_extra qnx -d -g
ENABLE_NATIVE_DP_SST_SAFETY=1 C2C_OPTION=rp_only -s 1
#./make/bind_partitions -b $BOARD_ID -p prod_debug_extra qnx -d -g
ENABLE_NATIVE_DP_SST_SAFETY=1 C2C_OPTION=ep_only -s 2
```

```
#DISP_SER_MODULE=MAX96776 (Samsung 4k) OR MAX96745 || ENABLE_NATIVE_DP_SST_SAFETY=1
else
echo " BINDING STANDARD ... "
cd $LOCAL_PDK/drive-foundation
./make/bind_partitions -b $BOARD_ID qnx clean
./make/bind_partitions -b $BOARD_ID qnx -d -g -s 1
fi
```



Note: Current bind_partition CMD already used in NVIDIA DRIVE OS 6.0 PDK documentation. For more information, refer to the Bind Options chapter under AV PCT Configuration.

The resulting boot chains are:

```
[partition]
name=A_qspi_chain
allocation_policy=sequential
filesystem_type=basic
size=0x840000
partition_attribute=0x40000002
virtual_storage_ivc_ch=0x83802762
sub_cfg_file=boot_chain_storage.cfg
[partition]
name=B_qspi_chain
allocation_policy=sequential
filesystem_type=basic
size=0x840000
partition_attribute=0x40000002
virtual_storage_ivc_ch=0x83802863
sub_cfg_file=boot_chain_storage.cfg
```

The virtual_storage_ivc_ch depends on the VSC server and may be different than above. If you would like to use a custom sub config file for the platform rather than the default boot_chain_storage.cfg for either chain A or chain B, then replace cfg file path of sub_cfg_file=boot_chain_storage.cfg in global_storage.cfg file to your config file and run the bind command again. For instance, edit to sub_cfg_file=/tmp/myconfig.cfg under name=A_qspi_chain partition and execute the bind command.

The resulting configuration is:

```
[partition]
name=A_qspi_chain
allocation_policy=sequential
filesystem_type=basic
size=0x840000
partition_attribute=0x40000002
virtual_storage_ivc_ch=0x83802762
sub_cfg_file=/tmp/myconfig.cfg
[partition]
name=B_qspi_chain
allocation_policy=sequential
filesystem_type=basic
size=0x840000
partition_attribute=0x40000002
```

```
virtual_storage_ivc_ch=0x83802863
sub_cfg_file=boot_chain_storage.cfg
```

Use an absolute path to the configuration file if it does not reside in the virtualization folder for the given target and platform. The file is not checked to be valid or that the path exists. If the path is invalid, an error is thrown during the flashing process.

4.2.1 Flashing Customization

This section provides guidance on customizing flashing.

- > Bootburn
- > Kernel
- > Prebuilt binaries
- > Offline binaries

Select Boot Chain Platform Configuration

Within the board support package (BSP) with Virtualization support the top-level configuration file `global_storage.cfg` defines the sub configuration file used to create the target BSP. Currently, this defaults to `boot_chain_storage.cfg`. As an example, if you run the bind command:

Usage

```
./make/bind_partitions [options] [pct name] [make target]
```

Required Options

```
BOARD=p3710-10-s05
OS=qnx
```

Options

List of all options:

```
./make/bind_partitions [options] [pct name] [make target]
```

This wrapper script calls `boot_blob.mk`.

Short Code	Long Name	Description
-b	--board	Specifies the board variant.
-c	--chip_variant	Specifies the chip variant.
-d	--gos_dtb	Builds GOS device tree blobs.
-e	--cpu_type	Type of CPU to spoof.

Short Code	Long Name	Description
-f	--tos_flavor	TOS type (tzvault or other variant).
-g	--bpmp_dtb	Builds BPMP device tree blobs.
-h	--help	Shows this help message.
-i	--ssv_test	Virtualization test variant.
-n	--nfs_boot	Boots up Linux Guest OS mounting rootfs through NFS with specify Ethernet interface.
-p	--pct_variant	Select specify PCT variant profile file in DRIVE AV/DRIVE_IX profile.
-r	--storage_size	Type of storage config to use (32gb, 64gb, no_ufs).
-s	--soc_id	Specifies SOC ID in GOS device tree and rebuilds GOS device tree.
-t	--tos_target	Specifies the TOS target.
-u	--ndas_usecase	Specifies the NDAS enable or EcoSystem/General usecases.
-v	--verbose	Verbose output.
-w	--power_profile	Specifies the power profile.
-x	--chain_a	Specifies chain-A config file.
-y	--chain_b	Specifies chain-B config file.
-z	--fskp_fuse_burning	Enables fskp partition in Linux configuration for fuse burning.
N/A	--oot_kernel	Builds DTB for OOT kernel.
N/A	--scl_arg	Specifies argument for SCL components.

[pct name]: Specifies the PCT name to use.

[make target]: make target.

Examples

```
if [[ "$QNX_PDK" == "qnx-safety" ]]; then
```

```

echo " BINDING SAFETY . . . "
sleep 5
cd $LOCAL_PDK/drive-foundation-safety
./make/bind_partitions -b $BOARD -p prod_debug_extra qnx clean
./make/bind_partitions -b $BOARD -p prod_debug_extra qnx -d -g
ENABLE_NATIVE_DP_SST_SAFETY=1 C2C_OPTION=rp_only -s 1
#./make/bind_partitions -b $BOARD_ID -p prod_debug_extra qnx -d -g
ENABLE_NATIVE_DP_SST_SAFETY=1 C2C_OPTION=ep_only -s 2

#DISP_SER_MODULE=MAX96776 (Samsung 4k) OR MAX96745 || ENABLE_NATIVE_DP_SST_SAFETY=1
else
echo " BINDING STANDARD . . . "
cd $LOCAL_PDK/drive-foundation
./make/bind_partitions -b $BOARD_ID qnx clean
./make/bind_partitions -b $BOARD_ID qnx -d -g -s 1
fi

```



Note: Current bind_partition CMD already used in NVIDIA DRIVE OS 6.0 PDK documentation. For more information, refer to the Bind Options chapter under AV PCT Configuration.

The resulting boot chains are:

```

[partition]
name=A_qspi_chain
allocation_policy=sequential
filesystem_type=basic
size=0x840000
partition_attribute=0x40000002
virtual_storage_ivc_ch=0x83802762
sub_cfg_file=boot_chain_storage.cfg
[partition]
name=B_qspi_chain
allocation_policy=sequential
filesystem_type=basic
size=0x840000
partition_attribute=0x40000002
virtual_storage_ivc_ch=0x83802863
sub_cfg_file=boot_chain_storage.cfg

```

The virtual_storage_ivc_ch depends on the VSC server and may be different than above. If you would like to use a custom sub config file for the platform rather than the default boot_chain_storage.cfg for either chain A or chain B, then replace cfg file path of sub_cfg_file=boot_chain_storage.cfg in global_storage.cfg file to your config file and run the bind command again. For instance, edit to sub_cfg_file=/tmp/myconfig.cfg under name=A_qspi_chain partition and execute the bind command.

The resulting configuration is:

```

[partition]
name=A_qspi_chain
allocation_policy=sequential
filesystem_type=basic
size=0x840000

```

```

partition_attribute=0x40000002
virtual_storage_ivc_ch=0x83802762
sub_cfg_file=/tmp/myconfig.cfg
[partition]
name=B_qspi_chain
allocation_policy=sequential
filesystem_type=basic
size=0x840000
partition_attribute=0x40000002
virtual_storage_ivc_ch=0x83802863
sub_cfg_file=boot_chain_storage.cfg

```

Use an absolute path to the configuration file if it does not reside in the virtualization folder for the given target and platform. The file is not checked to be valid or that the path exists. If the path is invalid, an error is thrown during the flashing process.

4.2.2 Bind Steps for NVIDIA Orin System on a Chip (SOC)

For information about how to bind an NVIDIA Orin system on a chip, see See the "AV PCT Configuration" topic in the *NVIDIA DRIVE OS Linux PDK Developer Guide*.

4.2.3 Bootburn Usage

Use the Bootburn utility to flash platform boards with the boot loader, kernel, and file system. The Bootburn utility is provided as part of the Foundation package.

4.2.3.1 Bootburn Usage



Note: You do not need to use sudo for the following commands:

- > bootburn.py
- > create_bsp_images.py
- > flash_bsp_images.py

However, if you choose to use sudo for one, use it consistently for the others. Do not switch between sudo/non-sudo usage.

Use the Bootburn utility to flash platform boards with the boot loader, kernel, and file system. The Bootburn utility is provided as part of the Foundation package.

Bootburn

Bootburn is a shell script that:

- > Boots the device in recovery mode with RAMDisk. Recovery Mode is where BootROM waits on recovery USB to receive data. All boot images are received from the host by

NVIDIA Orin over USB and stored in RAM. The system is booted using the data stored in RAM.

- > Signs or generates the images on the host.
- > Transfers the images to target using ADB.
- > Flashes the boards.

Bootburn automatically resizes the rootfs image on the target to optimize the available storage on the partition dedicated to it, either on eMMC or NOR.

Bootburn supports a host NFS share as the mount point for the target rootfs. This feature provides a convenient way for developers to copy files between the target and host for rapid development. In this case, the uncompressed target rootfs directory on the host is leveraged directly by the target.

Bootburn relies on the Foundation services to provide the flashing and boot loader functionality. You configure it with the CFG file rather than with Bootburn command-line parameters; Bootburn supports limited command line options.

Bootburn Command Usage

Run the bootburn.py script from a shell on your host Linux system by executing the command:

```
./tools/flashtools/bootburn/bootburn.py <options>
```

Options	Description
<p>-b <board_name> (lowercase b)</p>	<p>Provides the board name. Possible board names include:</p> <ul style="list-style-type: none"> > p3710-10-a01 > p3710-10-a01-f1 > p3663-a01 > p3663-a01-f1 > p4024-a00-a > p4024-a00-a-f1 > p4024-a00-b > p4024-a00-b-f1 > p3898-b00 <p>See Section 1.0 of the Release Notes for your release to confirm your board and revision information.</p> <p>For more information, see Supported Platforms/PCTs and SDK/PDK Packages under API Modules.</p>
<p>--board_config <board_name>.json</p>	<p>Can be used with bootburn.py, create_bsp_images.py, and flash_bsp_images.py board_name must include absolute path.</p>

Options	Description
--clean	Erases all of QSPI and eMMC.
-d <partition_name> <dtb_file> (lowercase d)	<p>Specifies the DTB file to flash and the target partition. The default file is defined in <code>BoardSetFilePathsAndDefaultValues</code>.</p> <p><code>bmp-fw-dtb</code> and <code>kernel-dtb</code> may be specified simultaneously.</p> <p>Note: This option is not intended to specify <code>kernel-dtb</code> in Hypervisor.</p>

Options	Description
<code>-h</code> (lowercase h)	Provides guidance on the actions and options of bootburn.
<code>-i (lowercase i)</code>	File path to platform configuration file. If not provided for hypervisor configurations, then the tool expects the platform configuration file to be present in the hypervisor output directory
<code>-k <configuration file></code> (lowercase k)	Specifies the configuration file for flashing.
<code>-m (lower case)</code>	Flashing Modular Diagnostic Software (MODS). This option enables mods-specific overrides.
<code>-o</code> (lowercase o)	<p>Skips flashing of recovery partitions. Used by minicom.</p> <p>WARNING: If you are using minicom with -o option, minicom does not initialize the port and therefore does NOT check or acquire the lock files for the communication port. If minicom is running in the background and you run bootburn with the -x option, bootburn acquires the lock BEFORE accessing the communication port. So, when bootburn attempts to communicate with AURIX it fails.</p>
<code>-p <key_file_path></code> (lowercase p)	<p>Signs the boot loader, secure OS (TOS), kernel and BCT binary, then flashes the device.</p> <p><code><key_file_path></code> must be the full pathname of the key_file.</p>
<code>--hsm <key_string></code>	<p>Can be used with bootburn.py and create_bsp_images.py. Tells Bootburn what keys will be used in HSM mode. Key String is Key + Encrypt Keys. Key is rsa eddsa. Encrypt Keys are sbk or kek0. Examples:</p> <ul style="list-style-type: none"> --hsm rsa --hsm rsa+sbk --hsm eddsa --hsm eddsa+sbk

Options	Description
-q (lowercase q)	Flashes QNX images.
-s (lowercase s)	Skips flashing the file system. Use the -s flag only when the target file system is already flashed.
-t <output_directory> (lowercase t)	Specifies the output directory where images to be flashed onto the target are to be stored. The default directory is named _temp_dump, and is present in the flashing directory.
-u <partition_name ...> (lowercase u)	Specifies one or more partitions to create, flash, or create and flash.

Options	Description
-x	<p>Specifies the communication port for AURIX to put the SoC in recovery mode automatically.</p> <p>With this -x option, bootburn performs target validation for the user provided board name on the command line, with the -b option, before flashing the board.</p> <p>Bootburn generates a unique ID from the target Inforom data. The allowed target ID list for the user provided board is generated from BOM data file included in the package. Bootburn aborts flashing if the target unique ID is not found in the list of generated unique IDs for the provided board name. This feature is supported on p3663 and p3710 boards with Inforom system object version 3 or above.</p> <p>Note: On boards with inforom, the bct sku info is updated with the information in the inforom.</p> <p>If the -x option is used during the AURIX setup, at the end during the AURIX reset of the flashing procedure there may be output messages of the form: "Killed sudo cat \$l_Aurix > \$p_FlashFiles/ \$l_AurixLogFile". These messages are not errors and can be safely ignored.</p> <p>WARNING: If minicom is used with the -o option, minicom does not initialize the port and therefore does NOT check or acquire the lock files for the communication port.</p> <ul style="list-style-type: none"> > If minicom is running in the background and you run bootburn with the -x option, bootburn acquires the lock BEFORE accessing the communication port. So when bootburn attempts to communicate with AURIX it fails. > If you quit minicom forcefully it may not fully stop the process and it will be lingering in the background. > When bootburn attempts to connect in this state it will acquire the port but the output will still be directed to the minicom running in the background. > Avoid using the -o option when using minicom.

Options	Description
-B <boot_device> (upper-case B)	Specifies the boot device, which must be either qspi or emmc. qspi is the default.
-C (upper-case C)	Specifies use of debug binaries for the boot loader.
-D (upper-case D)	Directs bootburn debug output to stdout along with the tool's regular output. If -D is not specified, debug output goes to a temporary file.
-E (upper-case E)	Enables DRAM ECC.
-I <bus_id> <device_id> (upper-case I)	<p>Flashes a specific device when multiple devices are in recovery.</p> <p>To get the bus and device ID of each device in recovery, enter the lsusb command on the host. For example, if lsusb gives the following output:</p> <pre>Bus 003 Device 105: ID 0955:7018 NVidia Corp. Bus 003 Device 104: ID 0955:7018 NVidia Corp.</pre> <p>Then flash the second device with the -I option as in this example:</p> <pre>-I 003 104</pre>
-L (upper-case L)	Enables low power modes. This option requires that the spe-fw and WBO firmware packages be included in the bootburn configuration because low-power modes require them.
-M (upper-case M)	Specifies development version firmware.

Options	Description
-R (upper-case R)	<p>Specifies RCM boot support, where the device boots without being flashed. When the target is in recovery mode, this option causes the flashing script to download all binaries, other than BCT, from the host. The target then reads the BCT from storage and other binaries from RAM. It boots the kernel directly on RAM.</p> <p>This option is much faster than when flashing also occurs. It is especially useful during debugging.</p> <p>Use the -R flag to initiate RCMBOOT with any of the commands mentioned in this table.</p>
-T (upper-case T)	<p>Enables tracing. Tracing logs are stored under the bootburn directory as log_trace*.txt.</p>
-U	Pass in the UFS provision configuration file.
-V (upper case V)	<p>Specifies read-back verification, where the binary images are read back (after writing) from the target's storage and compared with the original binary images. The two sets of images are compared to detect discrepancies.</p>
--customer-data	<p>Specified customer data such as skuinfo and others to be updated during flashing</p> <p>See flashing_customer_data.docx for information on how to use it.</p>
--init_persistent_partitions	Used to initialize persistent partitions. The image used to initialize the partition must be specified in the cfg file and the partition must be marked with ispersistent=yes.
--devicetype	Used to write single non-volatile memory device {spi, sdhci or ufshci}
--logs	Absolute path to directory for log files.

Kernel Parameter Combinations

Possible combinations of kernel parameters nfsdev, ip, and nfsroot are as follows. An error indicates that the initramfs shell is started.

nfsdev	ip	nfsroot	Result
(not set)	(not set)	(not set)	no network; root must be set in some other way
(not set)	(not set)	set	error; needs IP configuration
(not set)	set	(not set)	error; no root path to mount
(not set)	set	set	valid combination
set	(not set)	(not set)	valid combination; DHCP request is sent
set	(not set)	set	error; needs IP configuration
set	set	(not set)	error; no root path to mount
set	set	set	valid combination; no DHCP requests

Passing Additional Kernel Parameters

To pass additional kernel parameters, modify the os_args parameter of the kernel-dtb partitions in:

```
drive-<TBD_platform_ver>-foundation/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/<pct>/qnx_gos0_storage.cfg
```

Dialout Group

Add yourself to the dialout group since it is not possible to open the serial ports to the AURIX via sudo.

To add yourself to the dialout group:

```
sudo adduser <user name> dialout
```

You must log out and log back in for the command to take effect.

Bootburn Configuration File

The following is the list of keys that can be added to a JSON file and their meanings.

Name	Description	Default	Used By	Optional
f_NvQbDtblImage	Qb Dtb		Quick Boot	N

Name	Description	Default	Used By	Optional
f_MB1Pad	Pad Voltage/DPD cfg file	None	Tegrabct_v2	N
f_MB1Pinmux	Pinmux configuration File	None	Tegrabct_v2	N
f_MB1Prod	File containing Prod setting for pinmux	None	Tegrabct_v2	N
f_MB1Pmic	File containing PMIC-Real MMIO/I2C	None	Tegrabct_v2	N
f_MB1Misc	Misc Parameter File	None	Tegrabct_v2	N
f_MB1GpioInt	Gpio Interrupt Routing Cfg File	None	Tegrabct_v2	N
f_MB1BootDevice	Device specific Platform Cfg File	None	Tegrabct_v2	N
f_FlashingMB1SdRar	verified memory cfg from mem-qual (rcm-mode)	None	sw_memcfg_ov	N
f_MB1SdRamParam	verified memory cfg from mem-qual (Mission Mode)	None	sw_memcfg_ov	N
f_MB1wb0SdRamPa	verified memory cfg from mem-qual for SC7	None	sw_memcfg_ov	N
s_Kernel_Dtb_Varian	Substitution string for Kernel Dtb	None	Dtb Manipulations	If used
f_FlashingDtbdImg	RCM Flashing Dtb	None	Dtb Manipulations	N
f_BpmpFwDtb	Bpmp Fw Dtb	None	Dtb Manipulations	N
s_BPMP_Dtb_Variant	Substitution string for bpmp Dtb	None	Dtb Manipulations	If used
f_FlashingBpmpFwD	Flashing Bpmp Fw Dtb	None	Dtb Manipulations	N
f_DtbdImg	Guest Os Dtb	None	Dtb Manipulations	N
		None		N
f_UFSPhyLaneFile	Upphy lane cfg file (to extract USB Ian ID and ownership pair from)	None	Misc	N
s_FuseBypass	FuseByPass File Name	fuse_bypass	Misc	N
n_DramOverride	Does board support ECC	False	Bootburn scripts	Y

Name	Description	Default	Used By	Optional
s_ChipFamily	Chip family	None	Used by bootburn scripts	
s_Soc	SoC	None	Used by bootburn scripts	
s_chipID	Chip id	None	Used by bootburn scripts	
n_FlashFirmware	Flash firmware	False	Used by bootburn scripts	
N_FlashMicrosemi	Flash Microsemi	False	Used by bootburn scripts	
s_BrickOnTCFFlashin	Set to “True” to enable Atomic Flashing. TegraCoreFirmware(TCF) flashing. If all firmware is not flashed correctly, the target is scratched to make non-bootable.	False	Used by bootburn scripts	
s_Int	Pre-Prod – Soc Binning Value	None	Used by bootburn scripts.	
f_SignedCustomerData	Customer Data File in Board Configs	None	Used by bootburn scripts to fill in Br Bct Signed Customer Data.	

Example Board Configuration File

The following is an example of a JSON file. You can either use built-in path substitution or full paths names.

The built-in string substitutions are for PDK.

Name	Definition	Default	Value
p_PlatformBCT		None	<PdkTop>/drive-<TBD_platform_ver>-foundation/platform-config/hardware/nvidia/platform/t23x/automotive/bct
s_VerStr	Tegra Version	a01-	a01-
l_BootDevice	Boot Device	qspi	qspi, sdmmc, or ufs
s_DtblImgSuffix	Wrapper for Qnx	""	For Qnx “-qnxwrap”
p_BCT		None	<PdkTop>/drive-<TBD_platform_ver>-foundation/platform-config/bct/t23x
p_PlatformCommonBCT		None	<PdkTop>/drive-<TBD_platform_ver>-foundation/platform-config/hardware/nvidia/platform/t23x/common/bct

p3663-a01

```
{
    "p3663-a01":
    {
        "f_BootromCfg": "<p_PlatformBCT>/p3663/reset/tegra234-mb1-bct-p3663-a01-reset.dts",
        "f_BrBctDevParam": "<p_PlatformBCT>/common/bootrom/tegra234-br-bct-auto-<l_BootDevice>.dts",
        "f_NvQbDtbImage": "t23x-refs.dtb",
        "f_MB1Pad": "<p_PlatformBCT>/p3663/padvoltage/tegra23x-mb1-bct-p3663-a01-padvoltage-default.dts",
        "f_MB1Pinmux": "<p_PlatformBCT>/p3663/pinmux/tegra23x-mb1-bct-p3663-a01-pinmux.dts",
        "f_MB1Prod": "<p_PlatformBCT>/p3663/prod/tegra234-mb1-bct-p3663-a01-prod.dts",
        "f_MB1Pmic": "<p_PlatformBCT>/p3663/pmic/tegra234-mb1-bct-p3663-a01-pmic.dts",
        "f_MB1Misc": "<p_PlatformBCT>/p3663/misc/tegra234-mb1-bct-p3663-a01-misc.dts",
        "f_MB1Misc_Carveout": "<p_BCT>/misc/tegrabl_carveout_id.h",
        "f_MB2Bct": "<p_PlatformBCT>/p3663/misc/tegra234-mb2-bct-p3663-a01-auto.dts",
        "f_MB1VDK": "<NV_OUTDIR>/nvidia/bootloader/mb1-private/soc/t234/build/standard-boot/mb1_t234.bin",
        "f_FlashingMB1SdRamParam": "<p_PlatformBCT>/p3663/sdram/t234-p3663-a01-sdram.dts",
        "f_MB1SdRamParam": "<p_PlatformBCT>/p3663/sdram/t234-p3663-a01-sdram.dts",
        "f_MB1wb0SdRamParam": "<p_PlatformBCT>/p3663/sdram/t234-p3663-a01-sdram.dts",
    }
}
```

```

        "f_MB1GpioInt": "<p_PlatformBCT>/p3663/gpioint/tegra234-mb1-bct-p3663-a01-
gpioint.dts",
        "f_MB1BootDevice": "<p_PlatformBCT>/p3663/device/tegra234-mb1-bct-p3663-a01-
device.dts",
        "f_MB2Scr": "<p_PlatformBCT>/p3663/firewall/tegra234-mb2-bct-p3663-a01-
firewall.dts",
        "s_Kernel_Dtb_Variant": "p3663-0001-a01",
        "f_DtbImg": "tegra234-p3663-0001-a01-linux-native<s_DtbImgSuffix>.dtb",
        "f_FlashingDtbImg": "tegra234-p3663-0001-a01-flashing_base.dtb",
        "f_BpmpFwDtb": "tegra234-bpmp-3663-0001-a01.dtb",
        "s_BPMP_Dtb_Variant": "bpmp-3663-0001-a01",
        "f_FlashingBpmpFwDtb": "tegra234-bpmp-3663-0001-a01.dtb",
        "f_UFSPhyLaneFile": "<p_PlatformBCT>/p3663/uphy-lanes/tegra234-mb1-bct-p3663-a01-
uphylanе.dts",
        "s_FuseBypass": "fuse_bypass_t234.bin",
        "s_GrCsvString": "P3663",
        "s_ChipFamily": "t23x",
        "s_Soc": "t234",
        "s_chipID": "0x23",
        "s_Int": "F0",
        "f_SignedCustomerData": "p3663_signed_customer_data.json"
        "n_DramOverride": "True"
    }
}

```

Example of Successful Flash

Bootburn completed successfully!

Bootburn Time 505.16470408439636 seconds

```

returning to directory /media/home/t186-dev-main-1201/foundation/embedded/tools-t23x/
scripts/t234_bootburn_py
Cleaning up ...
    Cleaning temp dir

```

Example of Unsuccessful Flash

```

----- Stack Trace -----
stack frame 0 : PrintStackTrace(234) in /media/home/t186-dev-main-1201/foundation/
embedded/tools-t23x/scripts/t234_bootburn_py/flashtools_nvrror.py

stack frame 1 : AbnormalTermination(250) in /media/home/t186-dev-main-1201/foundation/
embedded/tools-t23x/scripts/t234_bootburn_py/flashtools_nvrror.py

stack frame 2 : GenerateImage(1791) in /media/home/t186-dev-main-1201/foundation/
embedded/tools-t23x/scripts/t234_bootburn_py/bootburn_lib.py

stack frame 3 : CreateFlashImages(1926) in /media/home/t186-dev-main-1201/foundation/
embedded/tools-t23x/scripts/t234_bootburn_py/bootburn_lib.py

stack frame 4 : bootburn_active(94) in ./bootburn.py

stack frame 5 : run(93) in /usr/lib/python3.4/multiprocessing/process.py

stack frame 6 : _bootstrap(254) in /usr/lib/python3.4/multiprocessing/process.py

```

```

stack frame 7 : _launch(77) in /usr/lib/python3.4/multiprocessing/popen_fork.py
stack frame 8 : __init__(21) in /usr/lib/python3.4/multiprocessing/popen_fork.py
stack frame 9 : _Popen(267) in /usr/lib/python3.4/multiprocessing/context.py
stack frame 10 : _Popen(212) in /usr/lib/python3.4/multiprocessing/context.py
stack frame 11 : start(105) in /usr/lib/python3.4/multiprocessing/process.py
stack frame 12 : bootburn(280) in ./bootburn.py
stack frame 13 : bootburnCommandLine(355) in ./bootburn.py
stack frame 14 : <module>(359) in ./bootburn.py
-----
s_ERROR_TOOL_NVIMAGEGEN

ERROR CODE = 53
s_ERROR_TOOL_NVIMAGEGEN
Exception raised in bootburn_active
Traceback (most recent call last):
  File "./bootburn.py", line 94, in bootburn_active
    bootburnLib.CreateFlashImages(targetConfig.f_FlashCfg)
  File "/media/home/t186-dev-main-1201/foundation/embedded/tools-t23x/scripts/t234_bootburn_py/bootburn_lib.py", line 1926, in CreateFlashImages
    self.GenerateImage(configFiles, l_Operation, p_TempDumpPath)
  File "/media/home/t186-dev-main-1201/foundation/embedded/tools-t23x/scripts/t234_bootburn_py/bootburn_lib.py", line 1791, in GenerateImage
    AbnormalTermination("s_ERROR_TOOL_NVIMAGEGEN", nverror.NvError_NvImagegen)
  File "/media/home/t186-dev-main-1201/foundation/embedded/tools-t23x/scripts/t234_bootburn_py/flashtools_nverror.py", line 258, in AbnormalTermination
    raise OSError(errorCode)
OSError: 53

returning to directory /media/home/t234-dev-main-1201/foundation/embedded/tools-t23x/scripts/t234_bootburn_py
Cleaning up ...
  Cleaning temp dir

```

Decoding Errors

Additional information is needed to fully determine the cause of some errors. That information can be found in the log files. The command executed determines the name of the log file (bootburn.txt, bootburnTegra-A.txt, BootburnTegra-B.txt, create_bsp.txt, or flash_bsp.txt). in the _t194_bootburn_py folder.

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_Success	0	Operation Worked	Consult log for more details.
NvError_NotImplemented	1	Context Specific	Consult log for more details.
NvError_NotSupported	2	Context Specific	Consult log for more details
NvError_NotInitialized	3	Incorrect setting of environmental variables	Consult log for more details
NvError_BadParameter	4	Either unknown parameter or incorrect use of options	Context Specific
NvError_Timeout	5	Timeout conditions waiting for a system event.	Consult log for more details.
NvError_InsufficientMemory	6	Context Specific	Consult log for more details.
NvError_ReadOnlyAttribute	7	Context Specific	Consult log for more details.
NvError_InvalidState	8	Context Specific	Consult log for more details.
NvError_InvalidAddress	9	Context Specific	Consult log for more details.
NvError_InvalidSize	10	bad Partition Size	Missing file or invalid cfg
NvError_BadValue	11	Operation Failed	Consult log for more details.
NvError_AlreadyAllocated	12	Context Specific	Consult log for more details.
NvError_Busy	13	Context Specific	Consult log for more details.
NvError_ModuleNotPresent	14	Context Specific	Consult log for more details.
NvError_ResourceError	15	Can't connect to Aurix Port	Consult log for more details

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_CountMismatch	16	Context Specific	Consult log for more details
NvError_OverFlow	17	Context Specific	Consult log for more details.
NvError_ImageCorrupt	18	Context Specific	Consult log for more details.
NvError_BadImage	19	Context Specific	Consult log for more details.
NvError_FuseBurningTemp	20	Context Specific	Consult log for more details
NvError_Mb1PartialUpdate	21	Context Specific	Consult log for more details
NvError_Sc7PartialUpdate	22	Context Specific	Consult log for more details
NvError_MtsPartialUpdate	23	Context Specific	Consult log for more details
NvError_Mb1OemSwRatch	24	Context Specific	Consult log for more details
NvError_MtsOemSwRatch	25	Context Specific	Consult log for more details
NvError_MtsRatchetScratch	26	Context Specific	Consult log for more details
NvError_Sc7OemSwRatch	27	Context Specific	Consult log for more details
NvError_Mb1Sc7OemSwRatch	28	Context Specific	Consult log for more details
NvError_HashMismatch	29	Context Specific	Consult log for more details
NvError_ImageMismatch	30	Context Specific	Consult log for more details
NvError_InvalidPtLayout	31	Context Specific	Consult log for more details
NvError_BCHNotCached	32	Context Specific	Consult log for more details

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_FuseBurningTemp	33	Context Specific	Consult log for more details
NvError_FuseBurningTemp	34	Context Specific	Consult log for more details
NvError_ECIDMisMatch	35	Context Specific	Consult log for more details
NvError_InActiveVinInvalid	36	Context Specific	Consult log for more details
NvError_BomDataBoardM	37	Board Bom Data File not Found	Missing Data File
NvError_TargetValidationF	38	Aurix Validation Failed	Consult log for more details
NvError_DevBinariesonPro	39	Context Specific	Consult log for more details
NvError_ProdBinariesonDe	40	Context Specific	Consult log for more details
NvError_TegraBctError	42	Call to Nvtetragabct_v2 failed	Consult log for more details
NvError_TegraRcmError	43	tegrarcm_v2 failed	Consult log for more details
NvError_TegraSkulInfoErro	44	Nvskulinfo return an error	Consult log for more details
NvError_TegraSignError	45	Tegrasign failed to sign	Bad key or missing file
NvError_NvImagegen	46	NvImagegen has failed	Consult log for more details
NvError_DeviceFailToBoot	47	Attempt to read Target uid failed	Consult log for more details
NvError_SystemCommand	48	Shell Command Failed	Consult log for more details
NvError_RatchetConfigNo	49	Ratchet Tool is not found	Invalid Configuration
NvError_AdbPull	50	Adb Pull Failed	Consult log for more details

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_AdbPush	51	Adb Push Failed	Consult log for more details
NvError_AdbShell	52	Target side shell script failed	Consult log for more details
NvError_Adb	53	Adb Operation Failed	Consult log for more details
NvError_Python	54	Context Specific	Consult log for more details
NvError_Bad_MD5	55	Md5 Checksum Failed	Device has corrupted data
NvError_InsufficientHostStorage	56	Not enough Storage Space	Can't create images
NvError_TegraParserError	57	TegraParser Failed	Error in call to TegraParser
NvError_TegraUfsProvisionFailed	58	Ufs Provision Failed	Consult log for more details
NvError_FileWriteFailed	59	Write Failed	Consult log for more details
NvError_FileReadFailed	60	Context Specific	Consult log for more details
NvError_EndOfFile	61	Context Specific	Consult log for more details
NvError_FileOperationFailed	63	Various Failures of Linux File Operations	Consult log for more details
NvError_OperationNotPermitted	64	Invalid Operation Attempted	Consult log for more details
NvError_DirOperationFailed	65	Missing Directory need	Consult log for more details
NvError_EndOfDirList	66	Context Specific	Consult log for more details
NvError_ConfigVarNotFound	67	Config Variable Not found	Consult log for more details
NvError_InvalidConfigVar	68	Sku Info not set or Low Power cfg issue	Consult log for more details

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_MemoryMapFailed	69	Context Specific	Consult log for more details
NvError_IoctlFailed	70	Aurix Validation Failed	Consult log for more details
NvError_AccessDenied	71	Aurix Validation Failed	Consult log for more details
NvError_DeviceNotFound	72	Requested Device can't be found	Device not found
NvError_KernelDriverNotFound	73	Aurix Validation Failed	Consult log for more details
NvError_FileNotFound	74	Missing File	Consult log for more details
NvError_InvalidArgument	75	Context Specific	Consult log for more details
NvError_ProcessNotFound	76	Context Specific	Consult log for more details
NvError_Deadlock	77	Context Specific	Consult log for more details
NvError_FileNameNotExist	78	Missing file	Consult log for more details
NvError_PartitionNotExist	79	Aurix Validation Failed	Consult log for more details
NvError_DeviceFailToRegister	80	Aurix Validation Failed	Consult log for more details
NvError_SystemCommandFailed	81	System Command Failed	Consult log for more details
NvError_CorruptedBuffer	82	Context Specific	Consult log for more details
NvError_SdioCardNotPresent	83	Context Specific	Consult log for more details
NvError_SdioInstanceTaken	84	Context Specific	Consult log for more details
NvError_SdioControllerBusy	85	Context Specific	Consult log for more details

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_SdioReadFailed	86	Context Specific	Consult log for more details
NvError_SdioWriteFailed	87	Context Specific	Consult log for more details
NvError_SdioBadBlockSize	88	Context Specific	Consult log for more details
NvError_SdioClockNotCor	89	Context Specific	Consult log for more details
NvError_SdioSdhcPattern	90	Context Specific	Consult log for more details
NvError_SdioCommandFa	91	Context Specific	Consult log for more details
NvError_SdioCardAlwaysP	92	Context Specific	Consult log for more details
NvError_SdioAutoDetectC	93	Context Specific	Consult log for more details
NvError_SdMmcRecoveryL	94	Context Specific	Consult log for more details
NvError_SdMmcTransferS	95	Context Specific	Consult log for more details
NvError_SdMmcStandByS	96	Context Specific	Consult log for more details
NvError_SdMmcIdleState	97	Context Specific	Consult log for more details
NvError_I2cReadFailed	98	Context Specific	Consult log for more details
NvError_I2cWriteFailed	99	Context Specific	Consult log for more details
NvError_I2cDeviceNotFou	100	Context Specific	Consult log for more details
NvError_I2cInternalError	101	Context Specific	Consult log for more details
NvError_I2cArbitrationFail	102	Context Specific	Consult log for more details

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_I2CCommunication	103	Context Specific	Consult log for more details
NvError_IdleHwError	104	Context Specific	Consult log for more details
NvError_IdleReadError	105	Context Specific	Consult log for more details
NvError_IdleWriteError	106	Context Specific	Consult log for more details
NvError_VsWriteError	107	Context Specific	Consult log for more details
NvError_VsReadError	108	Context Specific	Consult log for more details
NvError_ChipSkuClockLimit	109	Context Specific	Consult log for more details
NvError_ChipSkuNumCore	110	Context Specific	Consult log for more details
NvError_ChipSkuDmaFuse	111	Context Specific	Consult log for more details
NvError_ChipSkuNoTableE	112	Context Specific	Consult log for more details
NvError_ChipSkuInvalidDiv	113	Context Specific	Consult log for more details
NvError_MCMRevFyseRea	114	Context Specific	Consult log for more details
NvError_NvImageSign	115	Resign Failed	Consult log for more details
NvError_Unknown	116	Context Specific	Consult log for more details

USB Bus Access without sudo

sudo access is required to enable USB bus access. Edit or create `90-nvidia.rules` located at `/etc/udev/rules.d`. Add the line:

```
SUBSYSTEMS=="usb", ACTION=="add", ATTRS{idVendor}=="0955", MODE=="0666"
```

To reload the udev rules, run:

```
udevadm control --reload
```

4.2.3.1.1 Example Board Configuration File

The following is an example of a JSON file. You can either use built-in path substitution or full paths names.

The built-in string substitutions are for PDK.

Name	Definition	Default	Value
p_PlatformBCT		None	<PdkTop>/platform-config/hardware/nvidia/platform/t23x/automotive/bct
s_VerStr	Tegra Version	a02-	a01- or a02-
I_BootDevice	Boot Device	qspi	Qspi, sdmmc, or ufs
s_DtblImgSuffix	Wrapper for Qnx	""	For Qnx "-qnxwrap"
p_BCT		None	<PdkTop>/platform-config/bct/t23x
p_PlatformCommonBC		None	<PdkTop>/platform-config/hardware/nvidia/platform/t23x/common/bct

4.2.3.2 Kernel Parameter Combinations

Possible combinations of kernel parameters nfsdev, ip, and nfsroot are as follows. An error indicates that the initramfs shell is started.

nfsdev	ip	nfsroot	Result
(not set)	(not set)	(not set)	no network; root must be set in some other way
(not set)	(not set)	set	error; needs IP configuration
(not set)	set	(not set)	error; no root path to mount
(not set)	set	set	valid combination

nfsdev	ip	nfsroot	Result
set	(not set)	(not set)	valid combination; DHCP request is sent
set	(not set)	set	error; needs IP configuration
set	set	(not set)	error; no root path to mount
set	set	set	valid combination; no DHCP requests

4.2.3.2.1 Passing Additional Kernel Parameters

To pass additional kernel parameters, modify the `os_args` parameter of the `kernel-dtb` and `kernel-dtb-r` partitions in this file.

```
drive-foundation/virtualization/pct/<platform>/linux/linux_storage_qspi.cfg
```

Where `<platform>` is:

- > p3710-10-a01
- > p3710-10-a01-f1
- > p3663-a01
- > p3663-a01-f1
- > p4024-a00-a
- > p4024-a00-a-f1
- > p4024-a00-b
- > p4024-a00-b-f1

To bind:

```
cd <top>/drive-foundation ./make/bind_partitions linux -b <platform>
```

To flash all images for Linux:

```
cd <top>/drive-foundation/
tools/flashtools/bootburn_t23x
./bootburn.py -b <platform> -B qspi
```

To flash `kernel-dtb` only:

```
cd <top>/drive-<TBD_platform_ver>-foundation/tools/host/flashtools/bootburn_t19x
./bootburn.sh -b <platform> -B qspi -d kernel-dtb <kernel dtb filename>
```

4.2.3.3 RAM Spoofing for NDAS Use Case

For development use case running NDAS configuration it is possible to spoof 16 GB RAM configuration on a board having 32 GB of physical RAM.

This can be done using board_config bootburn option while flashing the board and using appropriate 16 GB spoofing board configuration file.

List of files board configuration file packaged for spoofing support:

```
p3710-10-a01_16GB_sdram.json  
p3710-10-a01-f1_16GB_sdram.json
```

For more information, see [Bootburn Options](#) for more details on –board_config options.

4.2.3.4 Example of Successful Flash

```
Bootburn completed successfully!  
  
[bootburn]: Bootburn Time 191.073220015 seconds  
  
returning to directory /localhome/drive-foundation-safety/tools/flashtools/bootburn  
Cleaning up ...  
    Cleaning temp dir
```

4.2.3.5 Example of Unsuccessful Flash

```
----- Stack Trace -----  
stack frame 0 : PrintStackTrace(234) in /media/home/t186-dev-main-1201/  
foundation/embedded/tools-t23x/scripts/t234_bootburn_py/flashtools_nverror.py  
  
stack frame 1 : AbnormalTermination(250) in /media/home/t186-dev-main-1201/  
foundation/embedded/tools-t23x/scripts/t234_bootburn_py/flashtools_nverror.py  
  
stack frame 2 : GenerateImage(1791) in /media/home/t186-dev-main-1201/  
foundation/embedded/tools-t23x/scripts/t234_bootburn_py/bootburn_lib.py  
  
stack frame 3 : CreateFlashImages(1926) in /media/home/t186-dev-main-1201/  
foundation/embedded/tools-t23x/scripts/t234_bootburn_py/bootburn_lib.py  
  
stack frame 4 : bootburn_active(94) in ./bootburn.py  
stack frame 5 : run(93) in /usr/lib/python3.4/multiprocessing/  
process.py  
  
stack frame 6 : _bootstrap(254) in /usr/lib/python3.4/  
multiprocessing/process.py  
  
stack frame 7 : _launch(77) in /usr/lib/python3.4/  
multiprocessing/popen_fork.py  
  
stack frame 8 : __init__(21) in /usr/lib/python3.4/  
multiprocessing/popen_fork.py  
  
stack frame 9 : _Popen(267) in /usr/lib/python3.4/  
multiprocessing/context.py  
  
stack frame 10 : _Popen(212) in /usr/lib/python3.4/  
multiprocessing/context.py
```

```

stack frame 11 : start(105) in /usr/lib/python3.4/
multiprocessing/process.py

stack frame 12 : bootburn(280) in ./bootburn.py

stack frame 13 : bootburnCommandLine(355) in ./bootburn.py

stack frame 14 : <module>(359) in ./bootburn.py

-----
s_ERROR_TOOL_NVIMAGEGEN
ERROR CODE = 53
s_ERROR_TOOL_NVIMAGEGEN
Exception raised in bootburn_active
Traceback (most recent call last):
  File "./bootburn.py", line 94, in bootburn_active
    bootburnLib.CreateFlashImages(targetConfig.f_FlashCfg)
  File "/media/home/t186-dev-main-1201/foundation/embedded/tools-t23x/
scripts/t234_bootburn_py/bootburn_lib.py", line 1926, in CreateFlashImages
    self.GenerateImage(configFiles, l_Operation, p_TempDumpPath)
  File "/media/home/t186-dev-main-1201/foundation/embedded/tools-t23x/
scripts/t234_bootburn_py/bootburn_lib.py", line 1791, in GenerateImage
    AbnormalTermination("s_ERROR_TOOL_NVIMAGEGEN", nverror.NvError_NvImagegen)
  File "/media/home/t186-dev-main-1201/foundation/embedded/tools-t23x/
scripts/t234_bootburn_py/flashtools_nverrror.py", line 258, in AbnormalTermination
    raise OSError(errorCode)
OSError: 53

returning to directory /media/home/t186-dev-main-1201/foundation/embedded/
tools-t23x/scripts/t234_bootburn_py
Cleaning up ...
    Cleaning temp dir

```

4.2.3.6 Decoding Errors

Additional information is needed to fully determine the cause of some errors. That information can be found in the log files. The command executed determines the name of the log file (bootburn.txt, bootburnTegra-A.txt, BootburnTegra-B.txt, create_bsp.txt, or flash_bsp.txt). in the _t23x_bootburn_py folder.

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_NotSupported	2	Context Specific	Consult log for more details

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_NotInitialized	3	Incorrect setting of environmental variables	Consult log for more details
NvError_BadParameter	4	Either unknown parameter or incorrect use of options	Context Specific
NvError_Timeout	5	Timeout conditions waiting for a system event	Consult log for more details
NvError_InvalidSize	10	bad Partition Size	Missing file or invalid cfg
NvError_ResourceError	15	Can't connect to Aurix Port	Consult log for more details
NvError_BomDataBoardMapFileNotFound	37	Board Bom Data File not Found	Missing Data File
NvError_TargetValidationFail	38	Aurix Validation Failed	Consult log for more details
NvError_TegraBctError	49	Call to Nvtetragrabct_v2 failed	Consult log for more details
NvError_TegraRcmError	50	tegrarcm_v2 failed	Consult log for more details
NvError_TegraSkuInfoError	51	Nvskuinfo return an error	Consult log for more details

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_TegraSignError	52	Tegrasign failed to sign	Bad key or missing file
NvError_ImageMismatch	53	Must use Debug if using Virtual Box	User error
NvError_NvImagegen	53	NvImagegen has failed	Consult log for more details
NvError_DeviceFailToBoot	54	Attempt to read Target uid failed	Consult log for more details
NvError_SystemCommand	55	Shell Command Failed	Consult log for more details
NvError_RatchetConfigNotFound	56	Ratchet Tool is not found	Invalid Configuration
NvError_AdbShell	59	Target side shell script failed	Consult log for more details
NvError_Adb	60	Adb Operation Failed	Consult log for more details
NvError_Bad_MD5	62	Md5 Checksum Failed	Device has corrupted data
NvError_FileOperationFailed	196,611	Various Failures of Linux File Operations	Consult log for more details
NvError_DirOperationFailed	196,612	Missing Directory need	Consult log for more details

Error Name	Numeric Value	Meaning	Most Likely Cause
NvError_ConfigVarNotFound	196,614	Missing file	Consult log for more details
NvError_InvalidConfigVar	196,615	Sku Info not set or Low Power cfg issue	Consult log for more details
NvError_DeviceNotFound	196,625	Requested Device can't be found	Device not found
NvError_FileNotFound	196,627	Missing File	Consult log for more details
NvError_InvalidArgument	196,628	Context Specific	Consult log for more details
NvError_FileNameNotExist	196,631	Missing file	Consult log for more details
NvError_SystemCommandFailed	196,634	System Command Failed	Consult log for more details

4.2.4 Updating Customer Data into BCT



Note: Customer data can only be specified at `create_bsp_images.py/flash_bsp_images.py` when the target has been fused for authentication. In this case, the customer data can only be updated when a privacy key is available.

There are two types of customer data:

1. Product specific : Unique to product
2. Target specific : Unique to target

Customer data is typically done with a read, modify, and write process, except for the authentication exception noted above. Two methods are provided are explained below.

Product Customer Data

Product specific customer data can be specified using the board json file. The following key/file is added to board specific json file.

```
"f_SignedCustomerData": "p3710_signed_customer_data.json"
```

This is intended for customer data items that never or rarely change.

Target Specific Customer Data

Target specific customer data can be specified at any time (creation or flashing).

4.2.4.1 Command Line Option

The command-line option is as follows.

Options	Description
--customer-data <json_data_file> [<json_schema_file>]	<p>Specifies customer data, such as skuinfo.</p> <p><json_data_file> specifies the customer data values in a JSON file.</p> <p><json_schema_file> specifies the optional customer data schema files.</p>



Note: Arguments -z and -A are no longer supported.,

	Note:
	Partial customer data updates are supported. It is not required to update all the data fields specified in the schema.

4.2.4.1.1 Usage

```
./bootburn.py --customer-data customer_data.json -b p3663-a01 -B qspi
./create_bsp_images.py --customer-data customer_data.json nv_customer_data_schema.json -
b p3663-a01 -r 02 -g ./flashing_images
./flash_bsp_images.py --customer-data customer_data.json -b p3663-a01 -P ./
flashing_images/940-0RIN-2200-000_VD*
```

4.2.4.2 Customer Data Schema File Format

```
{
  "<BCT Section Name>": {
    "allowed-offset-ranges": [[<Left range value>, <Right range value>], ...],
    "data": {
      "<Data field name>": {
        "parser-metadata": {
          "data-separator-string": Delimiter characters separated by "|",
          "number-of-words": number of words after separating the word
        },
        "schema": {
          "version": {
            "type": "unsigned-char",
            "format-string": "B",
            "offset": Offset for BCT customer data where version number will be stored
          },
          "value": [
            {
              "type": One of the following allowed types: "decimal", "char", "hex", "string", "blob"
              "format-string": One of the following format strings: "%I", "%H", "%c", "%s", "%Q"
              "offset": Offset for BCT customer data, where this value will be stored
              "min-len": Minimum required length in bytes, only applies for string and char formats,
              "max-len": maximum required length in bytes, only applies for string and char formats,
            },
            [
              {
                "type": One of the following allowed types: "decimal", "char", "hex", "string", "blob"
                "format-string": One of the following format strings: "%I", "%H", "%c", "%s", "%Q"
                "offset": Offset for BCT customer data, where this value will be stored
                "min-len": Minimum required length in bytes, only applies for string and char formats,
                "max-len": maximum required length in bytes, only applies for string and char formats,
              },
              {
                ...
              },
              ...
            ],
            ...
          ]
        }
      }
    }
  }
  ...
}
}
```

4.2.4.3 Definitions

- > **BCT section name:** Name of the BCT section where data needs to be updated in BCT.
 - Supported sections at present are: "customer-data-unsigned" or "customer-data-signed".
- > **allowed-offset-ranges:** Array of allowed offset ranges for a section type where each range is specified as an array of [start of the range, end of the range].
- > **Data field name:** Field name for the data.
- > **parser-metadata:** Metadata used to parse the user input value and convert them into list of words represented by their schema given in "schema".
 - **data-separator-string:** Separator characters to allow separating user input data into individual words.
 - **number-of-words:** Expected number of words after data-separator-string is applied to the user input.

- > **schema**: Schema to interpret input data words and convert input data to bytes.
 - **version**: Version for the data item (defaults to 1)
 - Version schema object is optional and only need to be specified for the data items that require version information.
 - **value**
 - List of schema objects in sequence to allow interpreting each word in user input value for the data field.

 **Note:**
If the multiple format is supported for each word, schema for a word can be a list of schema objects.

- Each schema object is of the following format:

Type

Type	Interpretation
"decimal"	Used for integer formats "I", "H" and "Q"
"char"	Used for character format "c"
"string"	Used for string format "s"

- format-string:

Format String	Interpretation
"I"	Four bytes unsigned integer
"H"	Two bytes unsigned short integer
"Q"	Eight bytes unsigned integer
"c"	One byte characters
"s"	Null terminated string

- **offset**: Offset where value needs to be stored in the BCT section
 - > Offset values must be within the allowed range specified by "allowed-offset-ranges" for each section.
- **min-len**
 - > Minimum required input value length
 - > Only required for "c" and "s" format types

- **max-len**
 - > Maximum required input value length.
 - > Only required for "c" and "s" format types.

4.2.4.4 Format of the Data Value File

```
{
    "<Field Name>": "<Field Value>"
    "<Field Name>": {
        "version": <Version Value>,
        "value": <Field Value>,
    }
}
```

4.2.4.4.1 Description

Field Name: Name of the data field as specified in the schema (must match the name specified in the schema).

Field Value: Value of the data field.

version: Version value for the data field format.

value: Value of the data field (same as field value above).

4.2.4.4.2 Example customer data file

```
{
    "customer-data-signed": {
        "boardSerial": "223145", "macInUseCount": 8, "macId0": "mac0 0x00044BAF6805 3",
        "macId1": "mac1 0x00044BAF6815 3", "macId2": "mac2 0x00044BAF6825 1",
        "macId3": "mac3 0x00044BAF6835 4", "macId6": "mac6 0x00044BAF6865 7",
        "macId5": "mac5 0x00044BAF6855 6", "macId4": "mac4 0x00044BAF6845 5",
        "macId7": "mac7 0x00044BAF6875 2" }
}
```

4.2.4.4.3 Example schema file

```
{
    "customer-data-unsigned": {
        "allowed-offset-ranges": [[0, 1024]],
        "data": {
            "boardSerial": {
                "parser-metadata" : {
                    "data-separator-string": null,
                    "number-of-words": 1
                },
                "schema": {
                    "version": {
                        "type": "unsigned-char",
                        "format-string": "B",

```

```
        "offset": 16
    },
    "value": [
        {
            "type": "decimal",
            "format-string": "Q",
            "offset": 20
        }
    ]
},
"macInUseCount": {
    "parser-metadata" : {
        "data-separator-string": null,
        "number-of-words": 1
    },
    "schema": {
        "value": [
            {
                "type": "unsigned-char",
                "format-string": "B",
                "offset": 34
            }
        ]
    }
},
"macId0": {
    "parser-metadata" : {
        "data-separator-string": " ",
        "number-of-words": 3
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 40,
                "max-len": 10,
                "min-len": 1
            },
            {
                "type": "hex",
                "format-string": "Q",
                "offset": 50,
                "max-len": 6,
                "min-len": 6
            },
            {
                "type": "unsigned-char",
                "format-string": "B",
                "offset": 56
            }
        ]
    }
}
```

```
},
"macId1": {
    "parser-metadata" : {
        "data-separator-string": " ",
        "number-of-words": 3
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 64,
                "max-len": 10,
                "min-len": 1
            },
            {
                "type": "hex",
                "format-string": "Q",
                "offset": 74,
                "max-len": 6,
                "min-len": 6
            },
            {
                "type": "unsigned-char",
                "format-string": "B",
                "offset": 80
            }
        ]
    }
},
"macId2": {
    "parser-metadata" : {
        "data-separator-string": " ",
        "number-of-words": 3
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 88,
                "max-len": 10,
                "min-len": 1
            },
            {
                "type": "hex",
                "format-string": "Q",
                "offset": 98,
                "max-len": 6,
                "min-len": 6
            },
            {
                "type": "unsigned-char",
                "format-string": "B",
                "offset": 104,
                "max-len": 1,
                "min-len": 1
            }
        ]
    }
}
```

```
        "offset": 104
    }
]
}
},
"macId3": {
    "parser-metadata" : {
        "data-separator-string": " ",
        "number-of-words": 3
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 112,
                "max-len": 10,
                "min-len": 1
            },
            {
                "type": "hex",
                "format-string": "Q",
                "offset": 122,
                "max-len": 6,
                "min-len": 6
            },
            {
                "type": "unsigned-char",
                "format-string": "B",
                "offset": 128
            }
        ]
    }
},
"macId4": {
    "parser-metadata" : {
        "data-separator-string": " ",
        "number-of-words": 3
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 136,
                "max-len": 10,
                "min-len": 1
            },
            {
                "type": "hex",
                "format-string": "Q",
                "offset": 146,
                "max-len": 6,
                "min-len": 6
            }
        ]
    }
}
```

```
        },
        {
            "type": "unsigned-char",
            "format-string": "B",
            "offset": 152
        }
    ]
}
},
"macId5": {
    "parser-metadata" : {
        "data-separator-string": " ",
        "number-of-words": 3
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 160,
                "max-len": 10,
                "min-len": 1
            },
            {
                "type": "hex",
                "format-string": "Q",
                "offset": 170,
                "max-len": 6,
                "min-len": 6
            },
            {
                "type": "unsigned-char",
                "format-string": "B",
                "offset": 176
            }
        ]
    }
},
"macId6": {
    "parser-metadata" : {
        "data-separator-string": " ",
        "number-of-words": 3
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 184,
                "max-len": 10,
                "min-len": 1
            },
            {
                "type": "hex",
                "format-string": "Q",
                "offset": 194,
                "max-len": 6,
                "min-len": 6
            }
        ]
    }
}
```

```
        "format-string": "Q",
        "offset": 194,
        "max-len": 6,
        "min-len": 6
    },
    {
        "type": "unsigned-char",
        "format-string": "B",
        "offset": 200
    }
]
}
},
"macId7": {
    "parser-metadata" : {
        "data-separator-string": " ",
        "number-of-words": 3
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 208,
                "max-len": 10,
                "min-len": 1
            },
            {
                "type": "hex",
                "format-string": "Q",
                "offset": 218,
                "max-len": 6,
                "min-len": 6
            },
            {
                "type": "unsigned-char",
                "format-string": "B",
                "offset": 224
            }
        ]
    }
},
"VIN": {
    "parser-metadata" : {
        "data-separator-string": null,
        "number-of-words": 1
    },
    "schema": {
        "value": [
            {
                "type": "char",
                "format-string": "c",
                "offset": 264,
                "max-len": 19,
                "min-len": 1
            }
        ]
    }
}
```

```
        "min-len": 17
    }
]
}
},
"bomId": {
    "parser-metadata" : {
        "data-separator-string": null,
        "number-of-words": 1
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 284,
                "max-len": 19,
                "min-len": 1
            }
        ]
    }
},
"machineName": {
    "parser-metadata" : {
        "data-separator-string": null,
        "number-of-words": 1
    },
    "schema": {
        "value": [
            {
                "type": "string",
                "format-string": "s",
                "offset": 304,
                "max-len": 19,
                "min-len": 1
            }
        ]
    }
}
}
```

4.2.5 Generating Flashing Binaries Offline



Note: You do not need to use sudo for the following commands:

- > bootburn.py
- > create_bsp_images.py
- > flash_bsp_images.py

However, if you choose to use sudo for one, use it consistently for the others. Do not switch between sudo/non-sudo usage.

Use the `create_bsp_images.py` script to generate flashing binaries offline when the target is not connected to the host using `create_bsp` tools. Flashing binaries are pre-processed binaries used by the `update_sample` update tool.

The `create_bsp` tool `create_bsp_images.py` requires `bootburn_lib.py` and `bootburn` helper files for execution. The script uses default paths as in `bootburn` for the target. Depending on the configuration file, it processes raw binaries for flashing; including adding and signing headers. The `create_bsp_images.py` image invokes tools such as `nvimagegen`, `nvtetragen`, `nvtegrabct_v2` internally, similar to `bootburn.py`, to process binaries.

The `create_bsp_images.py` script is available at:

```
<top>/drive-foundation/tools/flashtools/bootburn
```

Prerequisites

To enable bootburn tracing support, execute the command:

```
sudo apt-get install -f lockfile-progs
```

create_bsp_images.py Command Line Options

The table below shows the parameter list for `create_bsp_images.py`.

Options	Description
-B <boot_device>	Specify the boot-device. Supported boot devices are qspi and emmc. Default is qspi.
-C	Selects debug binaries for boot loaders.
-D	Enables debug prints from flashing script.
-E	Specifies ECC-enabled binaries for Native Only
-M	Specifies development version firmware for non-production board.
-X	Value dumps into the GR memory carveout.

Options	Description
-Y	Specifies the dt-overlay odm-data parameter.
-b <board_name>	<p>Specifies the board name:</p> <ul style="list-style-type: none"> > p3710-10-a01 > p3710-10-a01-f1 > p3663-a01 > p3663-a01-f1 > p4024-a00-a > p4024-a00-a-f1 > p4024-a00-b > p4024-a00-b-f1 > p3898-b00
-c	Selects a safety profile: SCE with safety firmware, or APE with camera.
-d <partition_name> <filename>	<p>Specifies a DTB file.</p> <p>The default file is defined in the BoardSetFilePathsAndDefaultValues function in bootburn_helper.sh.</p> <p>bpmp-fw-dtb and kernel-dtb may be specified simultaneously.</p> <p>Note: This option is not intended to specify kernel-dtb in Hypervisor.</p>
-g	Generates binaries at the specified path. Does not flash.
-h	Provides help on options for the create_bsp_images tool.
-k	<p>Specifies the absolute path of the configuration file to be used.</p> <p>If used with Hypervisor, the configuration file must have been created from bind_partitions and be located in the hypervisor output directory.</p> <ul style="list-style-type: none"> > The default value for DRIVE OS Linux: quickboot_qspi_linux.cfg > The default value for DRIVE OS QNX: Quickboot_qspi_qnx.cfg

Options	Description
-l (Lower case l)	Creates Linux images.
-p <pkc_file>	Specifies RSA or ECDSA key file <pkc_file> for signing images.
-q	Creates QNX images.
--encryption_key	Takes as parameter the full file path and name. Secure boot encryption key. Can be used with bootburn.sh and create_bsp_images.sh.
--hsm <key_string>	<p>Can be used with bootburn.sh and create_bsp_images.sh. Tells Bootburn what keys will be used in HSM mode. Key String is Key + Encrypt Keys. Key is rsa eddsa. Encrypt Keys are sbk or kek0. Examples:</p> <ul style="list-style-type: none"> --hsm rsa --hsm rsa+sbk -- --hsm eddsa --hsm eddsa+sbk --hsm sbk
-r	<p>Specifies the chip revision. Supported values are:</p> <ul style="list-style-type: none"> > 01 > 02 <p>This option must be used with the -b option.</p>
-s (Lower case s)	Skips creating the file system.
--asymmetric	Used to generate images for native chain in asymmetric mode.

Options	Description
--customer-data	Specified customer data such as skuinfo and others to be updated during flashing.
--chain [A B C]	For use by DRIVE Update to generate update packages.
encrypt	Used to encrypt binaries
init_persistent_partition	<p>If a partition has the attribute ispersistent=yes</p> <p>Then the partition has a initialization file created</p>
merge_chains	This is an option to be used with asymmetric boot chains for manufacturing

4.2.5.1 Processed Binaries Directory Structure

The `create_bsp` tools take the `-g` argument to specify the output directory path where processed binaries are generated. After executing `create_bsp` tool, a per SKU-based directory is created under the output directory.

For example, if `create_bsp` is executed to create binaries for `--skunum 699-62382-0010-100 --setskuversion AA` with `-g` images, a sub-directory is created under the output directory, images in this case, with a name generated by appending the `skunum` and `setskuversion`.

```
$ ls images
699-62382-0010-100_AA
```

Under the sub-directory, three sets of binaries are generated as flash-images: `rcm-boot` and `rcm-flash`.

These binaries are required at different phases during flashing.

```
$ ls images/699-62382-0010-100_AA/
flash-images  rcm-boot  rcm-flash
```

The generated sub-directories are as follows:

Sub-directory	Description
flash-images	Contains processed binaries flashed on the target storage medium. Also contains FileToFlash.txt, a summary file that contains information about binary names to flash at offsets for the final target flash structure. These binaries are used for the update tool update_sample.
rcm-boot	Contains the necessary binaries to boot the target without writing binaries to the medium.
rcm-flash	Contains binaries similar to rcm-boot for booting the Linux kernel on initramfs without writing binaries to the medium. These binaries are generated from a set of prebuilt binaries that are used for flashing.

4.2.5.2 Generating Binaries for Flashing Asymmetric Boot Chains


Note:

Asymmetric is a manufacturing process to test, fuse, and load production software on an Orin SoC. For more information, refer to sections 5.5 and 7.4.3 in the *NVIDIA DRIVE OS 6.0 Safety Developer Guide*.

4.2.5.2.1 Prerequisites

- > Linux+MODS and AV+Q packages are installed.
- > Install fskp_fuseburn_py-<release>.tgz.
- > Install Mods on PDK package (AV+L MODS) on a per board basis.
- > For AV+QNX: Install AV + QNX Safety
- > For AV+Linux : Install AV + Linux

4.2.5.2.2 Create Fskp Firmware

This step combines fskp_fuse.xml (OEM xml file) and fskp_t234.bin. fskp_t23x.key is a private key checked out from NVIDIA provided after a formal request.

```
cd drive-foundation/tools/flashtools/fuseburn
./fskp_fuseburn.py -c 0x23 -f fskp_fuse.xml -k fskp_t23x.key -g <top>/drive-foundation/
firmware/bin/t234/fskpboot/ -i 63 -B <board> -b
```

or for hsm:

```
./fskp_fuseburn.py c 0x23 -f fskp_fuse.xml -k -g <top/drive-foundation/firmware/bin/
t234/fskpboot/ -i 63 -B <board> -b -hsm fskp
```

4.2.5.2.3 Bind

Bind for Single Linux GOS VM+MODS

```
cd drive-foundation
bind_partitions -b <board> PCT=linux -z PCT_VARIANT=mods clean
bind_partitions -b <board> PCT=linux -z PCT_VARIANT=mods
```

4.2.5.2.4 How to Create and Merge Asymmetric Boot Images

Create A chain:

```
cd ${NV_WORKSPACE}/drive-foundation
tools/flashtools/bootburn/create_bsp_images.py -b <board> -r 1 -g ${PWD}/<board>/chain_a
-D --chain A --asymmetric --fskp-bct-path ${NV_WORKSPACE}/drive-foundation/firmware/bin/
t234/fskpboot/br_bct_BR_sigheader.bct -m
```

Create B chain:

- > Using a privacy key for all images

```
# Specify the key
${PWD}/tools/flashtools/bootburn/create_bsp_images.py -b <board> -r 1 -g ${PWD}/
<board>/chain_b -D --chain B --asymmetric --encryption_key <Path to encryption key
file> -p <Path_to_signing_key_file>
```

- > Using a unique privacy key per SoC

```
# Do not specify the key
${PWD}/tools/flashtools/bootburn/create_bsp_images.py -b <board> -r 1 -g ${PWD}/
<board>/chain_b -D --chain B --asymmetric --encryption_key <Path to encryption key
file>
cd ${NV_WORKSPACE}
# Merge chains
${NV_WORKSPACE}/drive-foundation/tools/flashtools/bootburn/create_bsp_images.py
-b <board> -r 1 -g ${NV_WORKSPACE}/<merge-chain> --asymmetric --merge-chains
A=<chain_a> B=<chain_b>
```

For example,

```
 ${NV_WORKSPACE}/drive-foundation/tools/flashtools/bootburn/create_bsp_images.py -b
 p3710-10-a04 -r 1 -g ${NV_WORKSPACE}/p3710-10-a04-merge -D --asymmetric --merge-
chains A=${NV_WORKSPACE}/drive-foundation/p3710-10-a04/chain_a B=${NV_WORKSPACE}/
drive-foundation-safety/p3710-10-a04/chain_b
```

- > Additional steps for using a unique key per SoC

1. Sign the base package with the new unique key.

```
 ${NV_WORKSPACE}/drive-foundation/tools/flashtools/bootburn_t23x_py/
post_processing_tool.py --chip 0x23 --images ${NV_WORKSPACE}/p3710-10-a04-
merge/642-63710-0010-000_TS4/flash-images/ --headers-output-dir ${NV_WORKSPACE}/
p3710-10-a04-headers --asymmetric --signing-key ~/keys/edopenssl_v3_0.pem --debug
```

2. Generate a new fuse block for the unique key (updated fskp_fuse.xml). For more
information, see [Create Fskp Firmware](#).

```
./fskp_fuseburn.py -c 0x23 -f fskp_fuse.xml -k fskp_t23x.key -g ${NV_WORKSPACE}/
drive-foundation/firmware/bin/t234/fskpboot/ -i 63 -B <board> -b
```

3. Copy the FSKP blob.

```
cp ${NV_WORKSPACE}/drive-foundation/firmware/bin/t234/fskboot/
blob_fskp_updated_aligned_sigheader_encrypt.signed ${NV_WORKSPACE}/p3710-10-a04-
headers
```

4.2.5.3 Output

Output images: Output is saved in the directory path specified via input JSON configuration file using `output-directory : directory path`.

Logs: Log files are generated for each `create_bsp_images.py` run using the `<name>.log` where `<name>` is the chain name given for each chain in the JSON input configuration file.

4.2.6 Flashing Preprocessed Binaries



Note: You do not need to use sudo for the following commands:

- > `bootburn.py`
- > `create_bsp_images.py`
- > `flash_bsp_images.py`

However, if you choose to use sudo for one, use it consistently for the others. Do not switch between sudo/non-sudo usage.

This topic describes how to flash prebuilt binaries that `create_bsp_images.py` generated. Such binaries are generated offline, when the target is disconnected from the host.

The flash BSP tool, `flash_bsp_image.py`, calls the following scripts:

- > `bootburn_lib.py`
- > `bootburn_adb.py`
- > `bootburn_helper.py`

Flash BSP flashes binaries that are generated offline by `create_bsp_tool.py`.

Flash BSP uses the following tools:

- > `tegrarcm_v2`
- > `adb`
- > `nvdd`
- > `nvskuinfo`

`flash_bsp_images.py` is at:

```
<top>/drive-
foundation/tools/flashtools/bootburn_t23x/
```

`flash_bsp_images.py` is at:

```
<top>/drive-
```

```
foundation/tools/flashtools/bootburn_t23x_py/
```

4.2.6.1 Usage

The options `flash_bsp_images.py` supports are as follows.

Option	Description
<code>-D</code>	Enables debug messages from the Flashing script.
<code>-I <bus_id> <device_id></code>	<p>Flashes a specific Orin when multiple devices are in recovery.</p> <p>Obtain the bus and device ID information of each NVIDIA device in recovery by executing the <code>lsusb</code> command on the host.</p> <p>For example, if the <code>lsusb</code> command gives the following output:</p> <pre>Bus 003 Device 105: ID 0955:7018 Nvidia Corp. Bus 003 Device 104: ID 0955:7018 Nvidia Corp.</pre> <p>Then flash the second NVIDIA device with the <code>-I</code> option as in the following example:</p> <pre>-I 003 104</pre>
<code>-P <path></code>	Specifies a directory from which <code>flash_bsp_images.py</code> picks prebuilt binaries. The script does not generate the binaries.
<code>-R</code>	Specifies RCM support. Boots without Flashing Software(BCT is used from Media). Default flash-boot
<code>-b <board_and_rev></code>	Specifies the board name.
<code>-h</code>	<p>Provides guidance on the <code>flash_bsp_images</code> tool options.</p> <p> Note:</p> <p><code>flash_bsp_images.py</code> supports only the options described in this table, which are a subset of the options listed by this <code>-h</code> option. (This <code>-h</code> option lists all options supported by <code>bootburn_lib.py</code>.)</p>
<code>-o</code>	Skips flashing of recovery partitions.

Option	Description
-u <partition_name ...>	Specifies one or more partition names to flash.
--asymmetric	Flag to specify asymmetric boot chain flashing.
--customer-data	Specified customer data such as skuinfo and others to be updated during flashing. See Updating Customer Data into BCT for information on how to use it.
devicetype	Used to write single non-volatile memory device {spi, sdhci or ufshci}.
logs	Absolute path to directory for log files.

4.2.6.2 Directory Structure of Preprocessed Binaries

The Flash BSP tools -P argument specifies the output directory path with the processed binaries.

The examples in this topic assume all images are in the `images` directory

For flashing, Flash BSP obtains the prebuilt images to flash from:

```
./images/<SKUInfo>-<SKUVersion>
```

For example:

```
$ ls images
```

The following subdirectories are present in the `images/<SKUInfo>` directory:

- > flash-images
- > rcm-boot
- > rcm-flash

For example:

```
$ ls images/699-63550-0001-300_GD/
flash-images  rcm-boot  rcm-flash
```

These directories contain different binaries that are used at different phases in flashing.

4.2.6.2.1 To flash the prebuilt binaries

1. Connect the target to host PC and put target in recovery.
2. Flash the binaries to the target. For example:

```
./flash_bsp_images.py -P images/699-63550-0001-500_AB
```

```
-b e3550b03-t194a
```

Because this command omits the -z option, the Flash BSP tool reads SKUInfo from the target and preserves it in the BCT.

4.2.6.3 Flashing the Prebuilt Binaries

To flash the prebuilt binaries:

1. Connect the target to host PC and put target in recovery.
2. Flash the binaries to the target.

For example, the following command can be used to flash Orin:

```
./flash_bsp_images.py -P images/699-63550-0001-500_AB -b <board>
```

Because this command omits the -z option, the Flash BSP tool reads SKUInfo from the target and preserves it in the BCT.

4.2.6.4 Flashing SKUInfo

The subdirectory name under the output directory, `images` in this example, is very important because it contains SKUInfo. Flash BSP parses SKUInfo to obtain platform detail.

Flash BSP tools also accepts the -z option with arguments to fill SKUInfo in BR-BCT file. The same BR_BCT file (with SKUInfo passed with -z) is updated to target.

4.2.6.4.1 To flash SKUInfo and the prebuilt binaries

1. Connect the target to host PC and put target in recovery.
2. Flash the binaries to the target. For example:

```
$ ./flash_bsp_images.py -P images/699-63550-0001-500_AB -b
e3550b03-t194a -z "--skunum 699-63550-0001-500 --setskuversion AB
--setboardserial 01234 --setprodinfo 699-63550-0001-500 AB
--setmacid mac0 0x000ba4eba5"
```



Note:

The original copy of BCT SKUInfo is modified after `flash_bsp.py` executes.

4.2.6.5 Flashing Asymmetric Boot Chain Images

You can flash an asymmetric chain by using the following command:

```
python3 ./flash_bsp_images.py -b <board> -P ${PWD}/<board>/<output_directory> --
asymmetric
```

If debugging information is required, specify the -D option.

For example,

```
 ${NV_WORKSPACE}/p3710-10-a04-merge/tools/flashtools/bootburn/flash_bsp_images.py -b p3710-10-a04 -D -P ${NV_WORKSPACE}/p3710-10-a04-merge/642-63710-0010-000_TS4/ --asymmetric
```

To flash an asymmetric BSP using a unique key per SoC, use the flashing command as follows:

```
 ./flash_bs_images.py -b <board> -P <output_directory> --headers <header-path>
```

For example,

```
 ${NV_WORKSPACE}/p3710-10-a04-merge/tools/flashtools/bootburn/flash_bsp_images.py -b p3710-10-a04 -D -P ${NV_WORKSPACE}/p3710-10-a04-merge/642-63710-0010-000_TS4/ --headers ${NV_WORKSPACE}/p3710-10-a04-fix-header/ --asymmetric
```

Where NV_WORKSPACE is the root directory that contains the BSP package and headers.

4.2.6.6 Flashing SKUInfo and Other Customer Data

Customer data, such as skuinfo and others, can be updated using the --customer-data option to `flash_bsp_images.py` when flashing.

See [Updating Customer Data into BCT](#) for details on how to use the --customer-data option.

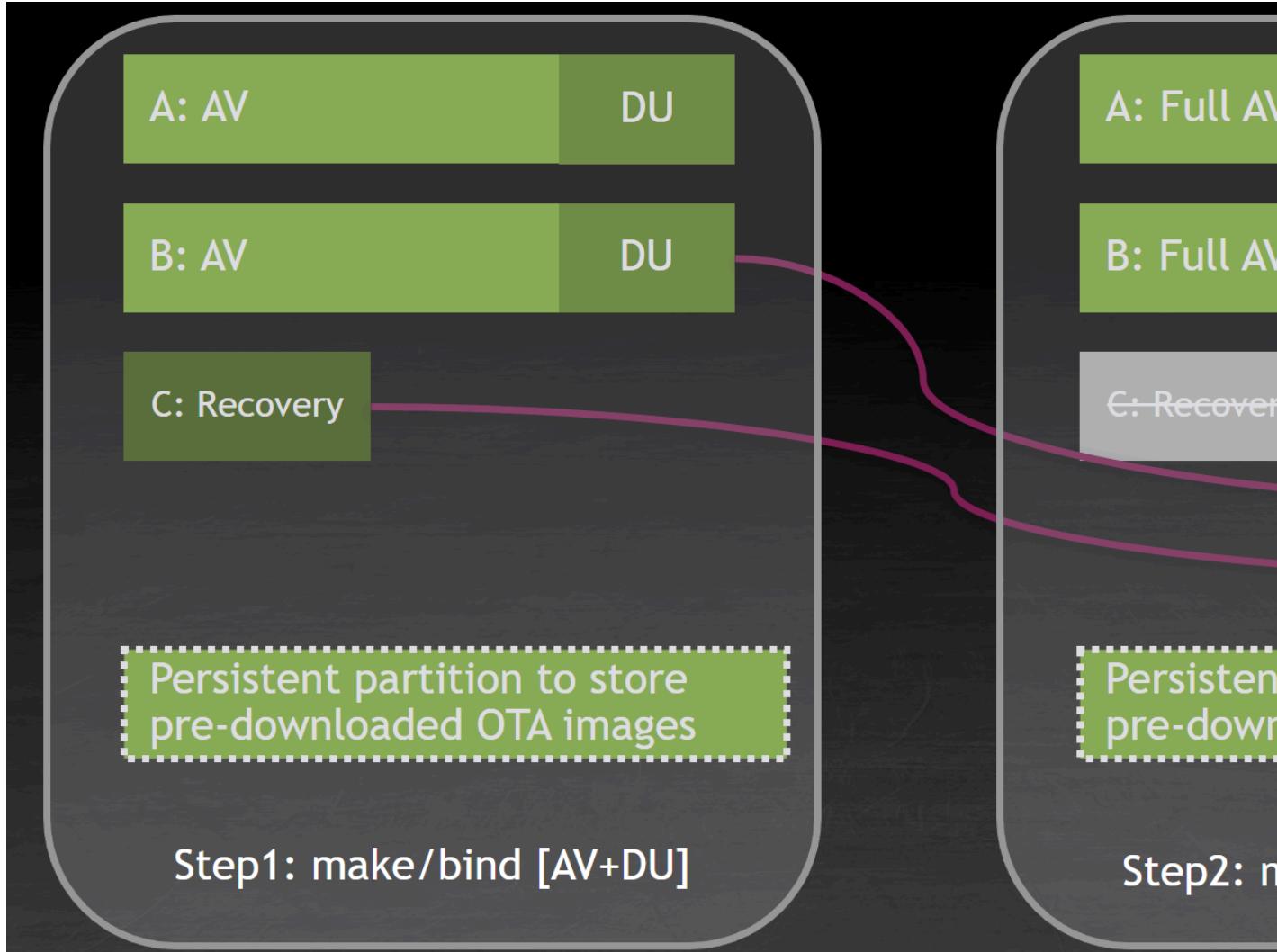
4.2.6.7 Flashing Flow

The flashing flow of flash BSP is similar to the bootburn flow except that flash BSP does not generate binaries.

4.2.7 Generating Image with DUVM Disabled in Chain A

Disabling DUVM in one bootchain saves about 512M memory but DUVM is required to support OTA, so DUVM must be enabled in another bootchain.

This chapter describes how to generate the image with DUVM disabled in chain A. The workflow is:



For example, on p3663-01 in Linux:

1. Create the DUVM enabled bsp_image:

```
cd drive-foundation
./make/bind_partitions --oot_kernel -b p3663-a01 linux
sudo rm -rf ${PWD}/../out/bsp_images_enable_duvm
./tools/flashtools/bootburn/create_bsp_images.py -b p3663-a01 -B qspi -r 1 -g
${PWD}/../out/bsp_images_enable_duvm
```

2. Create the DUVM disabled bsp_image:

```
cd drive-foundation
./make/bind_partitions --oot_kernel -b p3663-a01 linux ENABLE_UPDATE_VM=n
sudo rm -rf ${PWD}/../out/bsp_images_disable_duvm
./tools/flashtools/bootburn/create_bsp_images.py -b p3663-a01 -B qspi -r 1 -g
${PWD}/../out/bsp_images_disable_duvm
```

3. Combine the two bsp_image and merge the FileToFlash.txt:

```
/bin/cp -r ../out/bsp_images_enable_duvm ../out/bsp_images
/bin/cp -v ../out/bsp_images_disable_duvm/642-63663-0001-001_TS2/flash-images/A_* ../
out/bsp_images/*/flash-images/
```

Edit \${PWD}/../out/bsp_images/642-63663-0001-001_TS2/flash-images/FileToFlash.txt, remove all partitions starting with A_, and copy all the partitions starting with A_ from \${PWD}/../out/bsp_images_disable_duvm/642-63663-0001-001_TS2/flash-images/FileToFlash.txt.

For example:

```
${PWD}/../out/bsp_images/642-63663-0001-001_TS2/flash-images/FileToFlash.txt
```

```
# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize,
sku_dependent, BchPartitionName, ImageHeaderType, MD5, ReadWrite
/dev/block/3270000.spi bct A_bct_BR_zerosign.bct 0 524288 2 0 0 bct 11
0d834af29fe12c88cb4479749c2eee0b 1
/dev/block/3270000.spi pt 2_PT.bin 524288 524288 2 0 0 pt 12
38c394701d467caa22a79ff189d17ad6 1
/dev/block/3270000.spi bad-page 4_bad_page_zerosign.bin 47185920 524288 2 0 0 bad-
page 13 c8ae3b6fa7dd7f3edce3ef46c72e7dee 0
/dev/block/3270000.spi A_pt A_1_PT.bin 47710208 262144 1 0 0 A_pt 12
dbaf2e0e579b270df35884ed2b023303 0
/dev/block/3270000.spi A_mb1-bootloader A_2_mb1_t234_prod_zerosign.bin 47972352
524288 2 0 0 A_mb1-bootloader 12 8f9745dfa3bee493fb9acdf8ceae60e5 0
...
/dev/block/2500000.ufshci:0 A_2_kernel-dtb A_2_2_tegra_dtb_zerosign.dtb 29043720192
262144 64 0 0 A_2_kernel-dtb 12 6192fd847cac917e6c815a23dfdbe8d2 0
/dev/block/2500000.ufshci:0 A_2_kernel A_2_3_kernel_zerosign.img 29043982336 24641536
6016 0 0 A_2_kernel 12 cd311f5a0e155fb544c203845c29ae05 0
/dev/block/2500000.ufshci:0 pers-ota 12_pers-ota_null 111937585152 268435456 65536 0
0 pers-ota 4 d41d8cd98f00b204e9800998ecf8427e 1
# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize,
sku_dependent, BchPartitionName, ImageHeaderType, MD5, ReadWrite
/dev/block/3270000.spi B_pt B_1_PT.bin 57409536 262144 1 0 0 B_pt 12
dbaf2e0e579b270df35884ed2b023303 0
/dev/block/3270000.spi B_mb1-bootloader B_2_mb1_t234_prod_zerosign.bin 57671680
524288 2 0 0 B_mb1-bootloader 12 8f9745dfa3bee493fb9acdf8ceae60e5 0
...
/dev/block/2500000.ufshci:0 B_2_kernel-dtb B_2_2_tegra_dtb_zerosign.dtb 84878295040
262144 64 0 0 B_2_kernel-dtb 12 6192fd847cac917e6c815a23dfdbe8d2 0
/dev/block/2500000.ufshci:0 B_2_kernel B_2_3_kernel_zerosign.img 84878557184 24641536
6016 0 0 B_2_kernel 12 cd311f5a0e155fb544c203845c29ae05 0
# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize,
sku_dependent, BchPartitionName, ImageHeaderType, MD5, ReadWrite
/dev/block/3270000.spi C_pt C_1_PT.bin 1048576 262144 1 0 0 C_pt 12
e35f0efb6ceb1a9143f04a9b96839da0 0
/dev/block/3270000.spi C_mb1-bootloader C_2_mb1_t234_prod_zerosign.bin 1310720 524288
2 0 0 C_mb1-bootloader 12 8f9745dfa3bee493fb9acdf8ceae60e5 0
...
/dev/block/3270000.spi C_2_kernel-dtb C_2_2_tegra_dtb_zerosign.dtb 38535168 262144 1
0 0 C_2_kernel-dtb 12 91f6383654cd23d4c8e90a25836a2c7b 0
/dev/block/3270000.spi C_2_kernel C_2_3_kernel_zerosign.img 38797312 6291456 24 0 0
C_2_kernel 12 cd311f5a0e155fb544c203845c29ae05 0
```

```

${PWD}/..out/bsp_images_disable_duvm/642-63663-0001-001_TS2/flash-images/
FileToFlash.txt

# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize,
sku_dependent, BchPartitionName, ImageHeaderType, MD5, ReadWrite
/dev/block/3270000.spi bct A_bct_BR_zerosign.bct 0 524288 2 0 0 bct 11
0d834af29fe12c88cb4479749c2eee0b 1
/dev/block/3270000.spi pt 2_PT.bin 524288 524288 2 0 0 pt 12
0897c7557db1a5a9ac0cffbc9c434612 1
/dev/block/3270000.spi bad-page 4_bad_page_zerosign.bin 47185920 524288 2 0 0 bad-
page 13 c8ae3b6fa7dd7f3edce3ef46c72e7dee 0
/dev/block/3270000.spi A_pt A_1_PT.bin 47710208 262144 1 0 0 A_pt 12
986d803850e8b875b516d8e631341c9d 0
/dev/block/3270000.spi A_mb1-bootloader A_2_mb1_t234_prod_zerosign.bin 47972352
524288 2 0 0 A_mb1-bootloader 12 8f9745dfa3bee493fb9acdf8ceae60e5 0
...
...
/dev/block/2500000.ufshci:0 A_1_gos0-gpt A_1GPT_bak28_00.bin 27969581056 135168 33 0
0 A_1_gos0-gpt 4 99727eb540f7cf04721e2247947cb265 0
/dev/block/2500000.ufshci:0 A_gos0-rw-overlay A_gos0-rw-
overlay_35_ext4.img 27969716224 1073741824 262144 1 0 A_gos0-rw-overlay 4
a3a905f260af21eeec4fa816ff70a709 1
# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize,
sku_dependent, BchPartitionName, ImageHeaderType, MD5, ReadWrite
/dev/block/3270000.spi B_pt B_1_PT.bin 57409536 262144 1 0 0 B_pt 12
986d803850e8b875b516d8e631341c9d 0
/dev/block/3270000.spi B_mb1-bootloader B_2_mb1_t234_prod_zerosign.bin 57671680
524288 2 0 0 B_mb1-bootloader 12 8f9745dfa3bee493fb9acdf8ceae60e5 0
...
...
/dev/block/2500000.ufshci:0 B_1_gos0-gpt B_1GPT_bak28_00.bin 83804155904 135168 33 0
0 B_1_gos0-gpt 4 99727eb540f7cf04721e2247947cb265 0
/dev/block/2500000.ufshci:0 B_gos0-rw-overlay B_gos0-rw-
overlay_35_ext4.img 83804291072 1073741824 262144 1 0 B_gos0-rw-overlay 4
aa7089bb3a4bdd46ed51dd6b7dc1e448 1
# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize,
sku_dependent, BchPartitionName, ImageHeaderType, MD5, ReadWrite
/dev/block/3270000.spi C_pt C_1_PT.bin 1048576 262144 1 0 0 C_pt 12
e35f0efb6ceb1a9143f04a9b96839da0 0
/dev/block/3270000.spi C_mb1-bootloader C_2_mb1_t234_prod_zerosign.bin 1310720 524288
2 0 0 C_mb1-bootloader 12 8f9745dfa3bee493fb9acdf8ceae60e5 0
...
...
/dev/block/3270000.spi C_2_kernel-dtb C_2_2_tegra_dtb_zerosign.dtb 38535168 262144 1
0 0 C_2_kernel-dtb 12 2a4a2a367f01f64fda1fa3d1632c4156 0
/dev/block/3270000.spi C_2_kernel C_2_3_kernel_zerosign.img 38797312 6291456 24 0 0
C_2_kernel 12 cd311f5a0e155fb544c203845c29ae05 0

```

Remove the line A_pt to A_2_kernel in \${PWD}/..out/bsp_images/642-63663-0001-001_TS2/flash-images/FileToFlash.txt, then copy the A_pt to A_gos0-rw-overlay in \${PWD}/../out/bsp_images_disable_duvm/642-63663-0001-001_TS2/flash-images/FileToFlash.txt and paste to \${PWD}/..out/bsp_images/642-63663-0001-001_TS2/flash-images/FileToFlash.txt.

The final result in \${PWD}/../out/bsp_images/642-63663-0001-001_TS2/flash-images/FileToFlash.txt:

```
# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize,
sku_dependent, BchPartitionName, ImageHeaderType, MD5, ReadWrite
/dev/block/3270000.spi bct A_bct_BR_zerosign.bct 0 524288 2 0 0 bct 11
0d834af29fe12c88cb4479749c2eee0b 1
/dev/block/3270000.spi pt 2_PT.bin 524288 524288 2 0 0 pt 12
38c394701d467caa22a79ff189d17ad6 1
/dev/block/3270000.spi bad-page 4_bad_page_zerosign.bin 47185920 524288 2 0 0 bad-
page 13 c8ae3b6fa7dd7f3edce3ef46c72e7dee 0
/dev/block/3270000.spi A_pt A_1_PT.bin 47710208 262144 1 0 0 A_pt 12
986d803850e8b875b516d8e631341c9d 0
/dev/block/3270000.spi A_mb1-bootloader A_2_mb1_t234_prod_zerosign.bin 47972352
524288 2 0 0 A_mb1-bootloader 12 8f9745dfa3bee493fb9acdf8ceae60e5 0
...
...
/dev/block/2500000.ufshci:0 A_1_gos0-gpt A_1GPT_bak28_00.bin 27969581056 135168 33 0
0 A_1_gos0-gpt 4 99727eb540f7cf04721e2247947cb265 0
/dev/block/2500000.ufshci:0 A_gos0-rw-overlay A_gos0-rw-
overlay_35_ext4.img 27969716224 1073741824 262144 1 0 A_gos0-rw-overlay 4
a3a905f260af21eec4fa816ff70a709 1
/dev/block/2500000.ufshci:0 pers-ota 12_pers-ota_null 111937585152 268435456 65536 0
0 pers-ota 4 d41d8cd98f00b204e9800998ecf8427e 1
# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize,
sku_dependent, BchPartitionName, ImageHeaderType, MD5, ReadWrite
/dev/block/3270000.spi B_pt B_1_PT.bin 57409536 262144 1 0 0 B_pt 12
dbaf2e0e579b270df35884ed2b023303 0
/dev/block/3270000.spi B_mb1-bootloader B_2_mb1_t234_prod_zerosign.bin 57671680
524288 2 0 0 B_mb1-bootloader 12 8f9745dfa3bee493fb9acdf8ceae60e5 0
...
...
/dev/block/2500000.ufshci:0 B_2_kernel-dtb B_2_2_tegra_dtb_zerosign.dtb 84878295040
262144 64 0 0 B_2_kernel-dtb 12 6192fd847cac917e6c815a23dfdbe8d2 0
/dev/block/2500000.ufshci:0 B_2_kernel B_2_3_kernel_zerosign.img 84878557184 24641536
6016 0 0 B_2_kernel 12 cd311f5a0e155fb544c203845c29ae05 0
# LinuxPartitionName, PartitionName, FileName, Start, Size, BlockCount, Resize,
sku_dependent, BchPartitionName, ImageHeaderType, MD5, ReadWrite
/dev/block/3270000.spi C_pt C_1_PT.bin 1048576 262144 1 0 0 C_pt 12
e35f0efb6ceb1a9143f04a9b96839da0 0
/dev/block/3270000.spi C_mb1-bootloader C_2_mb1_t234_prod_zerosign.bin 1310720 524288
2 0 0 C_mb1-bootloader 12 8f9745dfa3bee493fb9acdf8ceae60e5 0
...
...
/dev/block/3270000.spi C_2_kernel-dtb C_2_2_tegra_dtb_zerosign.dtb 38535168 262144 1
0 0 C_2_kernel-dtb 12 91f6383654cd23d4c8e90a25836a2c7b 0
/dev/block/3270000.spi C_2_kernel C_2_3_kernel_zerosign.img 38797312 6291456 24 0 0
C_2_kernel 12 cd311f5a0e155fb544c203845c29ae05 0
```

Chapter 5. Supported Platform Configurations

5.1 AV PCT Configuration

Topics in this section:

5.1.1 Autonomous Vehicle Virtual Machine Configuration

The NVIDIA DRIVE AGX[™] system Autonomous Vehicle (AV) Partition Configuration Table (PCT) consists of server VMs, service VMs, and NVIDIA DRIVE AV Guest-OS (GOS) VM configurations.

PCT is divided according to the composition of the NVIDIA DRIVE AV GOS VM.

- Single Linux GOS VM
- Single QNX GOS VM
- Dual QNX GOS VMs

Profile Makefile

The profile makefile is the file containing the definitions of PCT configuration. Each PCT has its own profile makefiles.

Standard SDK/PDK Package

The default profile makefile (profile.mk) is for the standard package.

PCT name	PCT	Profile makefile for standard build
Single Linux GOS VM	linux	profile.mk
Single QNX GOS VM	qnx	profile.mk
Dual QNX GOS VMs	dual-qnx	profile.mk

The profile makefile is located at:

- > Single Linux GOS VM Standard (linux PCT):

```
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/linux/profile.mk
```

- > Single QNX GOS VM Standard (qnx PCT):

```
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/qnx/profile.mk
```

- > Dual QNX GOS VMs Standard (dual-qnx PCT):

```
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/dual-qnx/profile.mk
```

Linux Production PCT SDK/PDK Package

Linux Production build package needs additional option (PCT variant option, -p) when you use bind_partitions to select different profile makefile. The production package has two types of profile configurations (PCT variant): nsr_prod and nsr_prod_debug. For nsr_prod_debug, the PCT variant provides debug environment based on the nsr_prod PCT variant.

PCT name	PCT	PCT variant	Profile makefiles for production build	Comment
Single Linux GOS VM	linux	nsr_prod	profile_nsr_prod.mk	Supports DHCP in GOS VM.
		nsr_prod_debug	profile_nsr_prod_debug.mk	Combined UART is enabled. Servers/VM log available. Support SSH/DHCP/NFS in GOS VM.

- > Single Linux GOS VM Production (linux PCT with -p nsr_prod or -p nsr_prod_debug):

```
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/linux/profile_nsr_prod.mk
```

```
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/linux/profile_nsr_prod_debug.mk
```

QNX Safety PCT SDK/PDK Package

Safety build package needs additional option (PCT variant option, -p) when you use bind_partitions to select different profile makefile. The safety package has three types of profile configurations (PCT variant): prod, prod_debug, and prod_debug_extra. For prod_debug and prod_debug_extra, the PCT variant provides debug environment based on the prod PCT variant.

PCT name	PCT	PCT variant	Profile makefile for safety build	Comment
Single Linux GOS VM	linux	Not applicable	Not applicable	Linux PCT is not supported in the safety build.
Single QNX GOS VM	qnx	prod	profile_prod.mk	
		prod_debug	profile_prod_debug.mk	Supports communication to target over SSH/DHCP in GOS VM.
		prod_debug_extra	profile_prod_debug_extra.mk	Combined UART is enabled. Servers/VM log available. Supports SSH/DHCP/NFS in GOS VM.
Dual QNX GOS VMs	dual-qnx	prod	profile_prod.mk	
		prod_debug	profile_prod_debug.mk	Supports communication to target over SSH/DHCP in GOS VMs.
		prod_debug_extra	profile_prod_debug_extra.mk	Combined UART is enabled. Servers/VM log available. Supports SSH/DHCP/NFS in first GOS VM.

These profile makefiles are located at:

- Single QNX GOS VM Safety(qnx PCT with -p prod, -p prod_debug, or -p prod_debug_extra):

```
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/qnx/profile_prod.mk
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/qnx/profile_prod_debug.mk
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/qnx/profile_prod_debug_extra.mk
```

- > Dual QNX GOS VMs Safety(dual-qnx PCT with -p prod/prod_debug/prod_debug_extra):

```
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/dual-qnx/profile_prod.mk
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/dual-qnx/profile_prod_debug.mk
<top>/<NV_SDK_NAME_FOUNDATION>/platform-config/hardware/nvidia/platform/t23x/
automotive/pct/drive_av/dual-qnx/profile_prod_debug_extra.mk
```



Note: The prod_debug and prod_debug_extra PCT variants are used for testing and debugging purposes. These PCTs use a different file package that must not be used as part of the software stack in a running car.

Bootchain-C

During the bind_partitions process, chain-c profiles are automatically selected and Bootchain-C images are created depending on Platform/PCT. The following table shows platform support for Bootchain-C during boot-up.

platform	Support for Bootchain-C
P3663*	Yes
P3710-1*	Yes
P4024*	No
P3898*	No

The following table shows mapping between Bootchain-A/B PCT and Bootchain-C PCT profile.

Botochain-A B PCT	Botochain-A B PCT variant	Chain-C profile file (Chain-C is always Linux PCT)	Chain-C PCT variant
AV+L	none (standard)	linux/profile_chain_c.mk	none (standard)
	nsr_prod	linux/profile_chain_c.mk	none (standard)
	nsr_prod_debug	linux/profile_chain_c.mk	none (standard)
AV+Q	none (standard)	linux/profile_chain_c.mk	none (standard)

Botochain-A B PCT	Botochain-A B PCT variant	Chain-C profile file (Chain-C is always Linux PCT)	Chain-C PCT variant
	prod	linux/ profile_chain_c_srf_prod.r	prod
	prod_debug	linux/ profile_chain_c_srf_prod.r	srf_prod
	prod_debug_extra	linux/ profile_chain_c_srf_prod_	srf_prod_debug

Supported Platform and CPU Allocation

The following tables list the supported combinations of PCT, platform, and SDK/PDK package.

The *CPUs* column indicates the number of CPUs assigned to the guest VM and to the server VMs, respectively.

Supported Platform and Board Name

Official Name	Platform	Board Name	940/694- BOARD-SKU- REV	Comment
NVIDIA DRIVE Orin N™	p3898	p3898-b00	940-63898-0000-20	p3898 supports only chip sku TA977SA.
NVIDIA DRIVE Orin™	p3663	<ul style="list-style-type: none"> > p3663-a01 > p3663-a01-f1 	<ul style="list-style-type: none"> > 940-63663-0000-00 	
		<ul style="list-style-type: none"> > p3663-01-a02 > p3663-01-a02-f1 	<ul style="list-style-type: none"> > 940-63663-0000-00 	Emmc Size increased to 64 GB compared to 32 GB on p3663-a01.
		<ul style="list-style-type: none"> > p3663-02-a02 > p3663-02-a02-f1 	<ul style="list-style-type: none"> > 940-63663-0000-00 	MAX96981B display serializer with DSC on top of p3663-01-a02 boards.

Official Name	Platform	Board Name	940/694-BOARD-SKU-REV	Comment
NVIDIA DRIVE AGX Orin DevKit	p3710	<ul style="list-style-type: none"> > p3710-10-a01 > p3710-10-a01-f1 > p3710-10-a01-ct03 > p3710-10-a01-ct04 	<ul style="list-style-type: none"> > 940-63710-0010 TS1 > 940-63710-0010 TS2 > 940-63710-0010 A00 > 940-63710-0010 	<ul style="list-style-type: none"> > ct03 is for chip sku TA983SA support. > ct04 is for chip sku TA977SA support.
		<ul style="list-style-type: none"> > p3710-10-a03 > p3710-10-a03-f1 	<ul style="list-style-type: none"> > 940-63710-0010 TS3 > 940-63710-0010 B00 	<ul style="list-style-type: none"> > Audio codec is changed from ALC5658 to ALC5640. > Aurix power sequence update.
		<ul style="list-style-type: none"> > p3710-10-a04 > p3710-10-a04-f1 	<ul style="list-style-type: none"> > 940-63710-0010 TS4 > 940-63710-0010 C00 	
		<ul style="list-style-type: none"> > p3710-10-s05 > p3710-10-s05-f1 	<ul style="list-style-type: none"> > 940-63710-0010 TS5 > 940-63710-0010 D00 > 940-63710-0010 RC1 > 940-63710-0010 > 940-63710-0010 > 940-63710-0010 	
		<ul style="list-style-type: none"> > p3710-12-a03 > p3710-12-a03-f1 	<ul style="list-style-type: none"> > 940-63710-0012 TS3 > 940-63710-0012 B00 	GMSL out interconnect board delta compared to DisplayPort Interconnect Board on p3710-10-a03.

Official Name	Platform	Board Name	940/694-BOARD-SKU-REV	Comment
		<ul style="list-style-type: none"> > p3710-12-a04 > p3710-12-a04-f1 	<ul style="list-style-type: none"> > 940-63710-0010 TS4 > 940-63710-0010 C00 	GMSL out interconnect board delta compared to DisplayPort Interconnect Board on p3710-10-a04.
		<ul style="list-style-type: none"> > p3710-12-s05 > p3710-12-s05-f1 	<ul style="list-style-type: none"> > 940-63710-0012 TS5 > 940-63710-0012 D00 > 940-63710-0012 RC1 > 940-63710-0012 D00 > 694-63710-0012 D00 > 694-63710-0012 B00 	GMSL out interconnect board delta compared to DisplayPort Interconnect Board on p3710-10-s05.
DRIVE Recorder	p4024	<ul style="list-style-type: none"> > p4024-a00-a > p4024-a00-a-f1 > p4024-a00-b > p4024-a00-b-f1 	<ul style="list-style-type: none"> > 940-14024-0000 A00 > 940-14024-0000 TS1 > 940-14024-0000 TS2 > 940-14024-0000 TS3 > 940-14024-0000 B00 	



Note: Board name with the -f1 or -ct03 or -ct04 suffix has 8 CPU cores. Also, board name with the -f1 suffix is only for NVIDIA Internal Use. Normally, NVIDIA Orin has 12 CPU cores.

CPU Assignment

Orin type	CPU assignment
12 CPU cores	12 for GOS0, 1 shared CPU with HOST1X server and 1 shared CPU with Other Servers(and GOS1).
8 CPU cores	8 for GOS0, 1 shared CPU with HOST1X server and 1 shared CPU with Other Servers.

The example [guest_config.h](#) shows the mapping between the guest OS and services as well as their allocations.

Use Cases of NDAS/ECO, eMMC/UFS Secondary Boot Device

Depending on the use cases, storage configuration and peripheral assignment are different.

The following table lists the supported use cases for platforms.

Use cases	First boot device + secondary boot device	Platforms
Driving ECU (NDAS)	QSPI+eMMC	p3710
EcoSystem/General (ECO)	QSPI+eMMC	p3663 / p3710 / p3898
	QSPI+UFS	p3710
Recorder (REC)	QSPI+eMMC	p4024

ECID Read Access on Guest VMs

Read access to ECID can be provided to Guest VMs by configuring below in `guest_config.h` for a VM in PCT settings.

```
can_read_ecid_hash = 1
```

ECID is considered as a solution for the user to get a unique ID of the platform from the user's application.



Note: ECID read access is disabled for NVIDIA DRIVE AV PCTs for all VMs by default.

5.1.2 Bind Options

A bind process creates a hypervisor image that combines DTB/IFS/KERNEL of server VMs, Hypervisor kernel, and PCT.



Note: Starting in 6.0.7.0, bind option needs pyparsing version 3.0.9.

Syntax for `bind_partitions`:

```
# for standard package/linux nsr production package
$ cd drive-foundation/

# for safety
$ cd drive-foundation-safety/

$ ./make/bind_partitions [-b <board_name>] <pct> [-p <pct_variant>] [-u <use_case>] [-r <storage_config>] [options]
```

Example of Standard Linux PCT with ECO (QSPI+UFS) use case on NVIDIA DRIVE AGX Orin™ Devkit:

```
bind_partitions -b p3710-10-a01 linux
```



Note: The default configuration of p3710-1* is Standard ECO QSPI+UFS. If you want ECO QSPI+eMMC boot, use the -u eco option.

Example of Standard QNX PCT with NDAS use case on NVIDIA DRIVE AGX Orin™ DevKit:

```
bind_partitions -b p3710-10-a01 qnx -u ndas
```

The following tables list the supported bind options.

Supported Bind Options



Note: Get the full <board_name> from [Supported Platform and Board Name](#).

Use cases	bind command	-b <board_nam	<pct> -p <pct_variant	-u <use_case> (default)	-r <storage_co (default)	Comment
ECO	bind_partition	<ul style="list-style-type: none"> > -b p3663-a01 > -b p3663-a01-f1 > -b p3663-0?a0? > -b p3663-0?a0?-f1 	<ul style="list-style-type: none"> > linux > linux -p nsr_prod > linux -p nsr_prod_ > qnx > qnx -p prod > qnx -p prod_debu > qnx -p prod_debu 	<ul style="list-style-type: none"> > (eco) 	<ul style="list-style-type: none"> > (32g) 	

Use cases	bind command	-b <board_nam	<pct> -p <pct_variant	-u <use_case> (default)	-r <storage_co (default)	Comment
		<pre>> -b p3710-1? 0?* > -b p3710-1? 0?-f1</pre>	<pre>> linux > linux -p nsr_prod > linux -p nsr_prod_ > qnx > qnx -p prod > qnx -p prod_debug > qnx -p prod_debug</pre>	<pre>> (eco_ufs_l > -u eco</pre>	<pre>> (64g) > -r 32g > -r no_ufs</pre>	<p>The -r no_ufs option disables UFS storage, which is used as the secondary boot device on P3710 default ECO use case. Since UFS will be disabled by this option, EMMC need to be the secondary boot device. To make EMMC as the secondary boot device of P3710 for the above case, specify the -u eco option with the -r no_ufs option.</p>
		<pre>> -b p3898- b00</pre>	<pre>> linux</pre>	<pre>> (eco)</pre>	<pre>> (32g)</pre>	<p>p3898 supports chip sku TA977SA and EMMC as the secondary storage device.</p>

Use cases	bind command	-b <board_nam	<pct> -p <pct_variant	-u <use_case> (default)	-r <storage_co (default)	Comment
NDAS	bind_partitions	<ul style="list-style-type: none"> > -b p3710-1?-0?* > -b p3710-1?-0?-f1 	<ul style="list-style-type: none"> > linux > qnx > qnx -p prod > qnx -p prod_debug > qnx -p prod_debug 	<ul style="list-style-type: none"> > -u ndas 	<ul style="list-style-type: none"> > (high) > -r base 	With the BASE storage configuration option (-r base), UFS device is disabled.
		<ul style="list-style-type: none"> > -b p3710-1?-0?* 	<ul style="list-style-type: none"> > dual-qnx > dual-qnx -p prod > dual-qnx -p prod_debug > dual-qnx -p prod_debug 	<ul style="list-style-type: none"> > (ndas) 		dual-qnx supports only NDAS storage configuration.
REC	bind_partition	<ul style="list-style-type: none"> > -b p4024-a00-a > -b p4024-a00-a-f1 > -b p4024-a00-b > -b p4024-a00-b-f1 	<ul style="list-style-type: none"> > linux 	Not applicable	Not applicable	

Bind Options for Power Profile

Platforms	Standard, Linux Production, or Safety	Power Profile (default)
p3898	Standard	<ul style="list-style-type: none"> > (MAXP-A-977-D-02)
p3710-1?-?0?	ECO use case: Standard or Linux production	<ul style="list-style-type: none"> > (MAXP-A-990-D-04) > -w MAXP-A-990-S-02

Platforms	Standard, Linux Production, or Safety	Power Profile (default)
	NDAS use case: QNX Standard/safety	> (MaxP-A-990-S-10)
	Safety	> (MAXP-A-990-S-02)
p3710-10-a01-ct03	Standard, Linux production, or safety	> (MAXP-A-983-D02)
p3710-10-a01-ct04	Safety	> (MAXP-A-977-D02) > -w MAXP-A-977-D01
p3710-10-a01-ct04	Standard or Linux production	> (MAXP-A-977-D01) > -w MAXP-A-977-D02
p4024	Standard	> (MAXP-A-990-D-03)

Bind Options for SOC ID for C2C in GOS-DT

Platforms	SOC ID for C2C in GOS-DT	Comments
p3663 / p3710 / p4024	-s <SOC_ID:1~4294967295>	Specify SOC ID in GOS-DT and rebuild GOS device tree.

When the <SOC_ID> argument is specified with the -s option, SOC_IDENTIFICATION_VALUE is defined and the <SOC_ID> value will be set for soc_id in GOS-DT property.

```
#ifdef SOC_IDENTIFICATION_VALUE
soc_id = <SOC_IDENTIFICATION_VALUE>;
#else
soc_id = <1>;
#endif
```

DM-VERITY and Root Filesystem Permission for DRIVE AV Linux PCTs

To enable dm-verity, root filesystem partition permission should be read-only. Therefore, enabling dm-verity and read-write root filesystem cannot co-exist and will cause the bind_partition command to fail. The following table shows the default values and possible combination of the OS_ARGS_ENABLE_DM_VERITY value and the OS_ARGS_ROOT_MOUNT_PER value.

Platform and use cases	Extra option of OS_ARGS_ENABLE	Extra option of OS_ARGS_ROOT_MOUNT	OS_ARGS_ENABLE value	OS_ARGS_ROOT_MOUNT_PER value
P3663/P3710/ P3898 AV+L ECO	Default (or OS_ARGS_ENABLE_DM_VERITY)	Default (or OS_ARGS_ROOT_MOUNT)	0	rw
		OS_ARGS_ROOT_MOUNT	0	ro

Platform and use cases	Extra option of OS_ARGS_ENABLE_DM_VERITY	Extra option of OS_ARGS_ROOT_MOUNT_PER	OS_ARGS_ENABLE_DM_VERITY value	OS_ARGS_ROOT_MOUNT_PER value
	OS_ARGS_ENABLE_DM_VERITY=1 (or OS_ARGS_ROOT_MOUNT_PER=ro)	OS_ARGS_ROOT_MOUNT_PER=ro	Bind error due to DM-verity enabled and root filesystem have rw permission.	
P3710 AV+L NDAS (-u ndas) P4024 AV+L-REC (-b p4024-*)	Default (or OS_ARGS_ENABLE_DM_VERITY=0)	Default (or OS_ARGS_ROOT_MOUNT_PER=rw)	0	ro
		OS_ARGS_ROOT_MOUNT_PER=rw	0	rw
	OS_ARGS_ENABLE_DM_VERITY=1 (or OS_ARGS_ROOT_MOUNT_PER=ro)	OS_ARGS_ROOT_MOUNT_PER=ro	1	ro
		OS_ARGS_ROOT_MOUNT_PER=ro	Bind error due to DM-verity enabled and root filesystem have rw permission.	

For instance, to enable dm-verity, you can specify OS_ARGS_ENABLE_DM_VERITY=1 OS_ARGS_ROOT_MOUNT_PER=ro with the bind_partitions command regardless of the default values.

```
bind_partitions -b p3663-a01 linux OS_ARGS_ENABLE_DM_VERITY=1 OS_ARGS_ROOT_MOUNT_PER=ro
```

Use NVMe Storage Instead of UFS

The P3710 platform has a UFS storage device module in the M.2 Key M slot by default. P3710 has the default ECO profile as QSPI and UFS with UFS as a secondary boot device by using the -u eco_ufs_boot option. If you want to replace the UFS storage with NVME, the default ECO profile will not boot. To do so, you must bind the ECO profile with QSPI and eMMC (with eMMC as the secondary boot device) by using the -r no_ufs option for PCT, as shown in the following example:

```
bind_partitions -b p3710-10-a01 -u eco -r no_ufs
```

The preceding options cause BCT, BPMP-DTB, GOS-DTB, and PCT to disable UFS device and enable NVMe (for example, PCIe configuration, storage partition layout, and disabling smmu of UFS partitions).

See [Bind Options](#) for more information about the bind_partitions command.

5.1.3 AV PCT Input/Output Resource Assignment

The ownership of input and output peripherals is divided between the guest OS and the service VMs.

The following table details the type of access the guest OS or service VMs have for each I/O peripheral.

Table 4. QNX and Linux PCT Standard/Production Package Profile

Resources	Resources Shared	Update VM	Guest OS
DRAM	Yes	512 MB	~30 GB(32 GB RAM) / ~13 GB(16 GB RAM)
iGPU	Yes	Not applicable	Virtual
DLA	No	Not applicable	DLA0, DLA1
PVA	No	Not applicable	PVA0, PVA1
NvEnc / OFA	No	Not applicable	Assigned
VIC	No	Not applicable	Assigned
Display	No	Not applicable	Assigned
QSPI	Yes	Virtual	Not applicable
eMMC0 (32/64 GB)	Yes	Virtual	Virtual
UFS (265 GB)	Yes	Virtual	Virtual
1G Ethernet	No	Not applicable	Assigned
SPI Master	No	Not applicable	Assigned
SPI Slave	No	Not applicable	Assigned (Only for Linux PCT)
RCE (ISP, VI, MIPICAL, NVCSI, CSI lanes)	No	Not applicable	Assigned
I2C Master	No	Not applicable	Assigned
GPIO	No	Not applicable	Assigned
Tegra CAN	No	Not applicable	Assigned
NVDEC	No	Not applicable	Assigned
NVJPG	No	Not applicable	Assigned
10G Ethernet	No	Not applicable	Assigned

Resources	Resources Shared	Update VM	Guest OS
PCIe Controller [5,6] EP+RP for C2C	No	Not applicable	Assigned (Only for P3710)
PCIe Controller [2], UART [2] for Wifi/BT	No	Not applicable	Assigned
SE Engine	Yes	Virtual	Virtual
I2S, A2B/codec driver	No	Not applicable	Assigned
dGPU	No	Not applicable	Assigned

Table 5. Dual-QNX PCT Standard/Production Package Profile

Resources	Resources Shared?	Update VM	Guest OS	Guest OS1
DRAM	Yes	512 MB	~29 GB(32 GB RAM) / ~12 GB (16 GB RAM)	512 MB
iGPU	Yes	Not applicable	Virtual	Not applicable
DLA	No	Not applicable	DLA0, DLA1	Not applicable
PVA	No	Not applicable	PVA0, PVA1	Not applicable
NvEnc / OFA	No	Not applicable	Assigned	Not applicable
VIC	No	Not applicable	Assigned	Not applicable
Display	No	Not applicable	Assigned	Not applicable
QSPI	Yes	Virtual	Not applicable	Not applicable
eMMC0 (32/64 GB)	Yes	Virtual	Virtual	Virtual
UFS (265 GB)	Yes	Virtual	Virtual	Not applicable
1G Ethernet	No	Not applicable	Assigned	Not applicable
SPI Master	No	Not applicable	Assigned	Not applicable
SPI Slave	No	Not applicable	Assigned (Only for Linux PCT)	Not applicable

Resources	Resources Shared?	Update VM	Guest OS	Guest OS1
RCE (ISP, VI, MIPICAL, NVCSI, CSI lanes)	No	Not applicable	Assigned	Not applicable
I2C Master	No	Not applicable	Assigned	Not applicable
GPIO	No	Not applicable	Assigned	Not applicable
Tegra CAN	No	Not applicable	Assigned	Not applicable
NVDEC	No	Not applicable	Assigned	Not applicable
NVJPG	No	Not applicable	Assigned	Not applicable
10G Ethernet	No	Not applicable	Assigned	Not applicable
PCIe Controller [5,6] EP+RP for C2C	No	Not applicable	Assigned (Only for P3710)	Not applicable
PCIe Controller [2], UART [2] for Wifi/BT	No	Not applicable	Not applicable	Assigned
SE Engine	Yes	Virtual	Virtual	Virtual
I2S, A2B/codec driver	No	Not applicable	Assigned	Not applicable
dGPU	No	Not applicable	Assigned	Not applicable



Note: Regarding SDRAM usage, the amount of physical memory available for each virtual machine for virtual RAM is determined by a fixed (base) amount plus a dynamic (growth) amount. The value for the fixed amount is read directly from the PCT. The value for the dynamic amount is based on the amount of memory remaining after the fixed allocation and subsequently assigned to each virtual machine based on a per-VM growth factor. Higher growth factors result in a higher proportion of memory allocated to a virtual machine.

HDMI Configuration

To support HDMI hot plugging,

1. The dtsi node dp_aux_ch0_hpd_pm0 of the pinmux configuration must be manually changed from "dp" to "rsvd1" below in pinmux/tegra234-mb1-bct-gpio-p3701-0001.dtsi:

```
dp_aux_ch0_hpd_pm0 {
```

```
...  
    nvidia,function = "rsvd1";  
};
```



Note: The `dp_aux_ch0_hpd_pm0` node is automatically generated using the steps described in [Configuring Pinmux and GPIO](#).

2. Enable GPIO port M pin 0 GPIO pin, which is HDMI hotplug GPIO pin in `pinmux/tegra234-mb1-bct-pinmux-p3701-0001.dtsi` as shown below:

```
gpio_main_default: default {  
    gpio-input = <  
        ...  
        TEGRA234_MAIN_GPIO(M, 0)  
        ...  
    >;  
};
```

5.1.4 GPIO Ownership

The [guest_gpio_ownership.h](#) shows the GPIO ownership to a guest OS.

5.1.5 NVIDIA DRIVE AV Storage Configuration

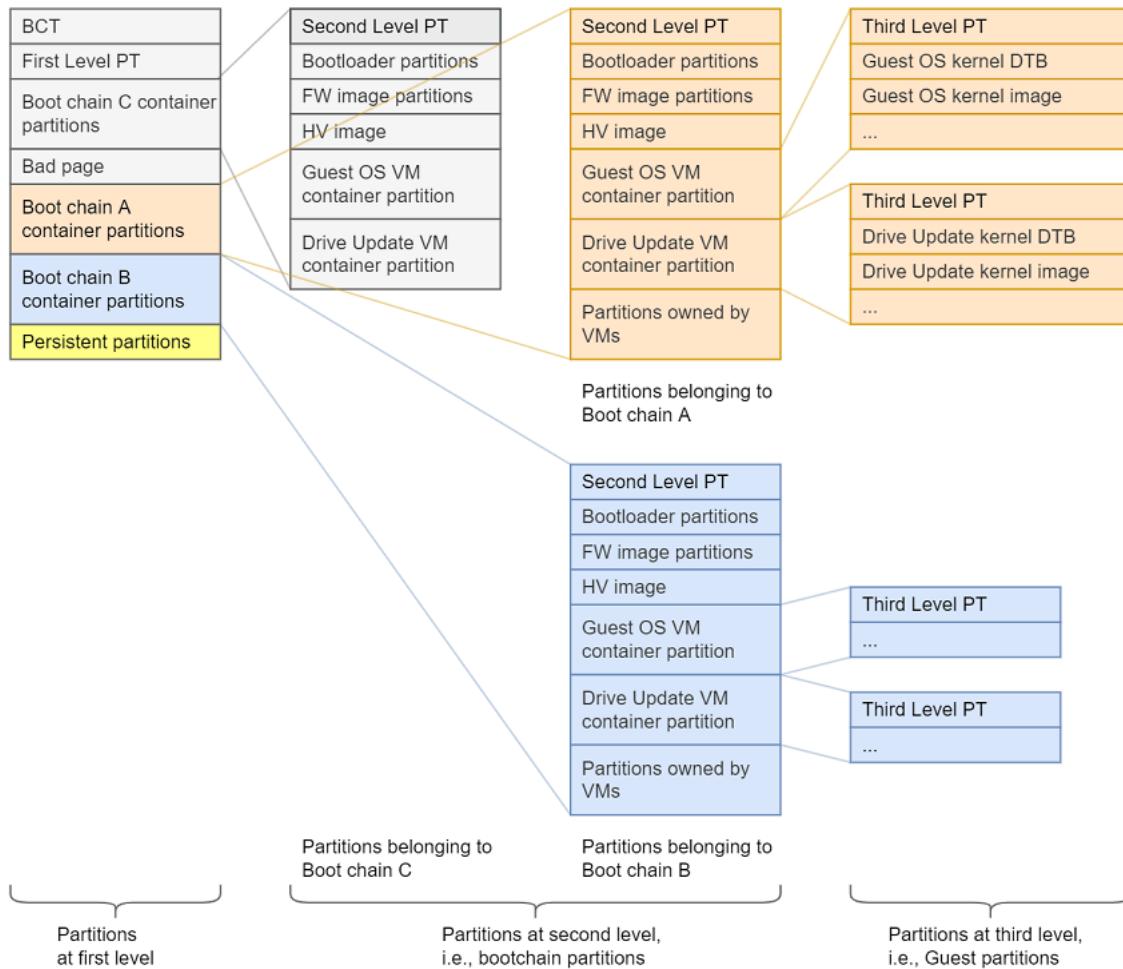
In the PCT, there are three configuration files that implement the three-level partitioning concept:

- > `global_storage.cfg`
 - This is the root of the definition hierarchy.
 - It defines the top-level partitioning and contains partitions that are present regardless of which boot chain is active.
 - It is referred to as the storage configuration file of the first-level (level-1) partition table.
- > `boot_chain_storage.cfg` file as mentioned after `sub_cfg_file=` in level-1 partitions.
 - Contains partitions that are unique to each of the boot chains.
 - It is referred to as the storage configuration file of the second-level (level-2) partition table.
- > `qnx_gos0_storage.cfg` file as mentioned after `sub_cfg_file=` in level-2 partitions.
 - Contains partitions that are unique to each of level-2 container partitions.
 - It is referred to as the storage configuration file of the third-level (level-3) partition table.

Three-Level Partitioning

The following diagram shows an example of a three-level partition layout and three-level partition tables.

Figure 1. Simplified View of Three-Level Configuration Files



To enable data updates in eMMC and UFS partitions through Over-The-Air(OTA), the partitions on eMMC are categorized as follows:

- > The eMMC A chain: It consists of all eMMC partitions assigned to all virtual machines in the Partition Configuration Table (PCT). All content in this chain can be overwritten by the OTA application.
- > The eMMC B chain: This is the recovery chain. The target boots in this chain when the OTA application is about to update other boot chains.
- > The QSPI C chain: This is the recovery chain to boot with only QSPI. The target boots in this chain when the OTA application is about to update other boot chains.
- > Persistent partitions for all virtual machines: The data partitions that are not updated by the OTA application. Data over these partitions remain consistent over multiple OTA cycles.

When inspecting the global_storage.cfg (level-1), you will notice it refers to boot_chain_storage.cfg as a sub_config file. This elevates the boot_chain_storage_qspi.cfg to be level-2.

```
...
[partition]
name=A_qspi_chain
...
sub_cfg_file=boot_chain_storage.cfg

[partition]
name=B_qspi_chain
...
sub_cfg_file=boot_chain_storage.cfg

[partition]
name=C_qspi_chain
...
sub_cfg_file=boot_chain_c_storage.cfg
```

Similarly, when inspecting the boot_chain_storage.cfg (level-2), you will notice it refers to the OS storage configuration file. This elevates the OS storage configuration file to be level-3.

```
[partition]
name=qnx-gos0
...
sub_cfg_file=qnx_gos0_storage.cfg
```

Since level-3 is derived from level-2, its content will be duplicated in each of the boot chains (A and B).

DRIVE AV Storage Layout

First-Level Partition Table (L1PT) Layout

The global_storage.cfg file defines the total size of storage devices, boot chain container partitions and persistent partitions of guest VMs.

The first-level configuration has the following layout for the ECO QSPI+eMMC use case:

Device	Partition	Size (Hex bytes)	IVC Value	Mandatory	Customizable	Purpose
QSPI - 0 size (0x4000000)	bct (BR-BCT)	0x80000	0x80162560	Yes	No	Holds board configuration information.
	pt (PT_1)	0x80000	0x80162661	Yes	No	L1 partition table.

Device	Partition	Size (Hex bytes)	IVC Value	Mandatory	Customizable	Purpose
	C_qspi_chain	0x2C00000	0x80962C66	No	Yes (Only related with GOS partitions)	L1 container partition for holding C chain on qspi.
	bad-page (PBL)	0x80000	N/A	Yes	No	For DRAM ECC.
	A_qspi_chain	0x9400000	0x80962762	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding A chain on qspi.
	B_qspi_chain	0x9400000	0x80962863	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding B chain on qspi.
EMMC - 3 size (0xE8FC00000 for 64GB) (0x747C00000 for 32GB)	A_emmc_chain	0x72D080000 for 64 GB 0x38D0C0000 for 32 GB	0x80962964	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding A chain on emmc.
	B_emmc_chain	0x72D080000 for 64 GB 0x38D0C0000 for 32 GB	0x80962A65	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding B chain on emmc.
	goso-shared-pers	0x10000000	0x801636ED	No	Yes	Persistent shared user partition of GOS0.
	pers-ota	0x10000000	0x8016245F	Yes	No	Persistent Storage for Update VM

Device	Partition	Size (Hex bytes)	IVC Value	Mandatory	Customizable	Purpose
UFS - 0 size (0x3A00000000)	A_ufs_chain	0xD00000000	0x80962D70	No	Yes (Only related with GOS partitions)	L1 container partition for holding A chain on ufs.
	B_ufs_chain	0xD00000000	0x80962E71	No	Yes (Only related with GOS partitions)	L1 container partition for holding B chain on ufs.
	gos0-ufs	0x1D0000000	0x80163FF7	No	Yes	User persistent data of GOS0.
	gos0-demo-ufs (only for linux)	0x280000000	0x801640F8	No	Yes	To demonstrate Encrypted File System feature.

The first-level configuration has the following layout for the ECO QSPI+UFS use case:

Device	Partition	Size (Hex bytes)	IVC Value	Mandatory	Customizable	Purpose
Device	Partition	Size (Hex bytes)	IVC Value (Depends on VSC Server GID)	Mandatory	Customizable	Purpose
QSPI - 0 size (0x4000000)	bct (BR-BCT)	0x80000	0x80162560	Yes	No	Holds Board configuration information.
	pt (PT_1)	0x80000	0x80162661	Yes	No	L1 Partition Table.
	C_qspi_chain	0x2C00000	0x80962C66	No	Yes (Only related with GOS partitions)	L1 container partition for holding C chain on qspi.
	bad-page (PBL)	0x80000	N/A	Yes	No	For DRAM ECC.

Device	Partition	Size (Hex bytes)	IVC Value	Mandatory	Customizable	Purpose
	A_qspi_chain	0x940000	0x80962762	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding A chain on qspi.
	B_qspi_chain	0x940000	0x80962863	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding B chain on qspi.
EMMC - 3 size (0xE8FC00000 for 64GB) (0x747C00000 for 32GB)	A_emmc_chain	0x72D080000 for 64 GB 0x38D0C0000 for 32 GB	0x80962964	Yes (For IST partitions)	Yes	L1 container partition for holding A chain on emmc.
	B_emmc_chain	0x72D080000 for 64 GB 0x38D0C0000 for 32 GB	0x80962A65	Yes (For IST partitions)	Yes	L1 container partition for holding B chain on emmc.
UFS - 0 size (0x3A00000000)	A_ufs_chain	0xD00000000	0x80962D70	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding A chain on ufs.
	B_ufs_chain	0xD00000000	0x80962E71	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding B chain on ufs.
	gos0-shared-pers	0x10000000	0x801636ED	No	Yes	Persistent shared user partition of GOS0

Device	Partition	Size (Hex bytes)	IVC Value	Mandatory	Customizable	Purpose
	pers-ota	0x10000000	0x8016245F	Yes	No	Persistent Storage for Update VM
	gos0-ufs	0x1D0000000	0x80163FF7	No	Yes	User persistent data of GOS0
	gos0-demo-ufs (only for linux)	0x280000000	0x801640F8	No	Yes	To demonstrate Encrypted File System feature

The first-level configuration has the following layout for the NDAS use case:

Device	Partition	Size (Hex bytes)	IVC Value	Mandatory	Customizable	Purpose
QSPI - 0 size (0x4000000)	bct (BR-BCT)	0x80000	0x80162560	Yes	No	Holds Board configuration information.
	pt (PT_1)	0x80000	0x80162661	Yes	No	L1 Partition Table
	C_qspi_chain	0x2C00000	0x80962C66	No	Yes (Only related with GOS partitions)	L1 container partition for holding C chain on qspi
	bad-page (PBL)	0x80000	N/A	Yes	No	For DRAM ECC
	A_qspi_chain	0x9400000	0x80962762	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding A chain on qspi
	B_qspi_chain	0x9400000	0x80962863	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding B chain on qspi

EMMC - 3 size (0x747C00000)	A_emmc_chain	0x500000000	0x80962964	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding A chain on emmc
	B_emmc_chain	0x500000000	0x80962A65	Yes	Yes (Only related with GOS partitions)	L1 container partition for holding B chain on emmc
	gos0-misc-pers	0x6600000	0x801235EC	Yes	Yes	
	gos0-ota-pers	0x10000000	0x801636ED	Yes	No	
	guest0-shadow-pers	0x66600000	0x809737EE	Yes	Yes	
	gos0-m-cache-pers	0x140000000	0x801238EF	Yes	Yes	
	gos0-m-stream-pers	0x400000000	0x801239F0	Yes	Yes	
	gos0-p-pers	0x200000000	0x80123AF1	Yes	Yes	
	gos0-s-logger-pers	0x600000000	0x80163BF2	Yes	Yes	
	gos0-sar-pers	0x133000000	0x800E3CF3	Yes	Yes	
	gos0-dlb-pers	0x40000000	0x80063DF4	Yes	Yes	
	gos1-config-pers	0xA00000 (Only for dual-qnx)	0x80962F72	No	Yes	
	pers-ota	0x100000000	0x8016245F	Yes	No	Persistent Storage for Update VM

UFS - 0 UFS device is only for HIGH storage configuration. size (0x1D00000000)	A_ufs_chain	0xC80000000	0x80962D70	No	Yes	L1 container partition for holding A chain on ufs
	B_ufs_chain	0xC80000000	0x80962E71	No	Yes	L1 container partition for holding B chain on ufs
	gos0-m- cache-ufs	0x200000000	0x801240F8	No	Yes	
	gos0-sar- ufs	0xC0000000	0x800E41F9	No	Yes	
	gos0-edr- ufs	0xC0000000	0x800A42FA	No	Yes	



Note: For mandatory partition, only the size of partition is customizable. For non-mandatory partition, all partition attributes are customizable or they can be removed or split.

Second-Level Partition Table (L2PT) Layout

> QSPI Chain-A/B Layout

QSPI partitions include bct files, bootloaders, and firmware binaries.

The storage layout for QSPI is as follows:

QSPI Partition Name	Size (in KBytes)	Mandatory	Customizable	Purpose
PT_2(pt)	256	Yes	No	L2 Partition Table
mb1-bootloader	512	Yes	No	Primary Copy of MB1 Bootloader
PSC-BL1 (psc-bl)	256	Yes	No	PSC firmware
MB1-BCT (mb1-bct)	256	Yes	No	BCT for MB1
MemBct (mem-bct)	256	Yes	No	BCT for memory configuration
IST_UCode(ccplex-ist-ucode)	256	Yes	No	IST ucode
MB2+MB2-BCT (mb2-bootloader)	512	Yes	No	MB2

QSPI Partition Name	Size (in KBytes)	Mandatory	Customizable	Purpose
SPE_FW (spe-fw) (Only for standard/safety prod_debug_extra PCT variant)	512	Yes	No	Sensor Processing Engine firmware
TSEC_FW (tsec-fw)	256	Yes	No	TSEC firmware
PSC_FW (psc-fw)	768	Yes	No	Firmware for PSC
MCE (mts-mce)	256	Yes	No	Firmware for cpu cores
BPMP_FW(bpmp-fw)	1536	Yes	No	Firmware for BPMP
SC7-RF	256	Yes	No	BPMP SC7 resume firmware
PSC-RF	256	Yes	No	PSC resume firmware
MB2-RF	256	Yes	No	CPU resume firmware
BPMP_FW_DTB (bpmp-fw-dtb)	512	Yes	No	DT for BPMP
RCE_FW (rce-fw)	1024	Yes	No	RCE firmware image
nvdec-fw	512	Yes	No	NVDEC firmware
Key IST uCode (ist-uicode)	256	Yes	No	IST key
IST_BPMP (bpmp-ist)	256	Yes	No	IST bpmp
IST_ICT (ist-config)	256	Yes	No	IST ICT
tsec-fw (tsec-fw)	256	Yes	No	TSEC FW

**Note:**

- > Chain B (same as Chain A) partitions are not included in the preceding table.
- > QSPI Chain-C Layout

QSPI Partition Name	Size (in KBytes)	Mandatory	Customizable	Purpose
PT_2(pt)	256	Yes	No	L2 Partition Table
mb1-bootloader	512	Yes	No	Primary Copy of MB1 Bootloader
PSC-BL1 (psc-bl)	256	Yes	No	PSC firmware
MB1-BCT (mb1-bct)	256	Yes	No	BCT for MB1
MemBct (mem-bct)	256	Yes	No	BCT for memory configuration
MB2+MB2-BCT (mb2-bootloader)	512	Yes	No	MB2
SPE_FW (spe-fw)	512	Yes	No	Sensor Processing Engine firmware
PSC_FW (psc-fw)	768	Yes	No	Firmware for PSC
MCE (mts-mce)	256	Yes	No	Firmware for cpu cores
BPMP_FW (bpmp-fw)	1536	Yes	No	Firmware for BPMP
BPMP_FW_DTB (bpmp-fw-dtb)	512	Yes	No	DT for BPMP
secure-os	4096	Yes	No	TOS
pvit	256	Yes	No	Partitions Version Info Table
fsi-fw	6144	Yes	No	FSI FW
kernel (HV image)	6656	Yes	No	HV kernel + server VMs + PCT
guest-linux-chain_c (Linux GOS VM 3LPT)	13568	Yes	Yes	Linux kernel/ initramfs/DT/GPT/ rootf
qnx-update-chain_c (IFS/DT of DRIVE Update VM L3PT)	6656	Yes	No	Update VM QNX primary IFS image and DT

> eMMC and UFS partitions

The eMMC/UFS device connected to Orin is partitioned logically to enable each VM to have its own root file system or to store user data. Each VM is assigned a dedicated user partition on eMMC/UFS and might have secondary user partitions for storing additional data. The eMMC/UFS device on board is shared between the VMs.

The eMMC/UFS storage chain A/B for QNX PCT ECO use case:

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
ist-testimg (For safety, Optional for standard)	1280	Yes	No	IST test image	eMMC
ist-runtimeinfo (For safety, Optional for standard)	0.25	Yes	No	IST runtime information	eMMC
ist-resultdata (For safety, Optional for standard)	200	Yes	No	IST result data	eMMC
gr-ist (For safety, Optional for standard)	0.25	Yes	No	gr blob support	eMMC
oist-tst-vtr	512	Yes	No	Runtime IST test vector	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
secure-os	4	Yes	No	TOS	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
adsp-fw (only for standard)	2	Yes	No	For Audio	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
fsi-fw (For safety, Optional for standard)	6	Yes	No	FSI FW	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
xusb-fw (Only for standard)	0.25	Yes	No	XUSB FW	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
pvit	0.25	Yes	No	Partitions Version Info Table	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
pva-fw	2.5	Yes	No	PVA FW	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
kernel (HV image)	10	Yes	No	HV kernel + server VMs + PCT	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
gos0-debug_overlay (For safety prod_debug*)	128	Yes	No	Debug overly for safety images only	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
qnx-gos0 (IFS/ DT for GOS0 L3PT)	31	Yes	Yes	GOS0 QNX primary IFS image and DT	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
gos0-ifs2 (Secondary IFS)	500	Yes	Yes	QNX secondary IFS image	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
usr-fs (GOS EFS1)	2560 (for standard) 2030 (for safety)	Yes	Yes	Guest OS rootfs	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
gos0-efs2 (GOS EFS2)	7680	No	Yes	Guest OS extended file system #2	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
gos0-efs3 (GOS EFS3)	1536 (for 32GB eMMC) 16384 (for 64GB eMMC or QSPI+UFS)	No	Yes	Guest OS extended file system #3	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
qnx-update (For IFS/DT of DRIVE Update VM L3PT)	24	Yes	No	Update VM QNX primary IFS image and DT	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
qnx-update-fs	128	Yes	No	Filesystem of Update VM QNX	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)

Where:

- > `/dev/vblk_*` denotes the enumeration for a given partition from guest OS or service VM. The partitions must be formatted before use.
- > IFS: Early boot partition with minimal file system that contains the kernel.
- > EFS: QNX root file system, additional file system demonstration bits, sample applications, and data.
- > All partitions present in `boot_chain_storage.cfg` are part of the OTA update.
- > All sub-configuration files that are in the `boot_chain_storage.cfg` are also part of the OTA update.

The eMMC/UFS storage chain A/B for Linux PCT ECO use case:

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
ist-testing (For safety, Optional for standard)	1280	Yes	No	IST test image	eMMC
ist-runtimeinfo (For safety, Optional for standard)	0.25	Yes	No	IST runtime information	eMMC
ist-resultdata (For safety, Optional for standard)	200	Yes	No	IST result data	eMMC

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
gr-ist (For safety, Optional for standard)	0.25	Yes	No	gr blob support	eMMC
gos0-crashlogs	1	No	Yes	To store Oops logs	eMMC
secure-os	4	Yes	No	TOS	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
adsp-fw	2	Yes	No	For Audio	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
fsi-fw	6	No	No	FSI FW	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
xusb-fw	0.25	Yes	No	XUSB FW	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
dce-fw	9	Yes	No	DCE FW	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
pvit	0.25	Yes	No	Partitions Version Info Table	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
pva-fw	2.5	Yes	No	PVA FW	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
kernel (HV image)	10	Yes	No	HV kernel + server VMs + PCT	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
guest-linux (For Linux GOS VM 3LPT)	11794 (for 32GB eMMC) 26642 (for 64GB eMMC or QSPI+UFS)	Yes	Yes	Linux kernel/ initramfs/DT/ GPT/rootfs	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
gos0-rw-overlay	1024	No	Yes	R/W partition for Read-only Rootfs	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
qnx-update (For IFS/DT of DRIVE Update VM L3PT)	24	Yes	No	Update VM QNX primary IFS image and DT	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)
qnx-update-fs	128	Yes	No	Filesystem of Update VM QNX	eMMC (QSPI +eMMC boot) UFS (QSPI +UFS boot)

The eMMC/UFS storage chain A/B for QNX (qnx/dual-qnx) PCT NDAS use case:

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
ist-testimg (For safety, Optional for standard)	1280	Yes	No	IST test image	eMMC
ist-runtimeinfo (For safety, Optional for standard)	0.25	Yes	No	IST runtime information	eMMC
ist-resultdata (For safety, Optional for standard)	200	Yes	No	IST result data	eMMC
gr-ist (For safety, Optional for standard)	0.25	Yes	No	gr blob support	eMMC
oist-tst-vtr	512	Yes	No	Runtime IST test vector	eMMC

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
secure-os	4	Yes	No	TOS	eMMC
adsp-fw (only for standard)	2	Yes	No	For Audio	eMMC
fsi-fw (For safety, Optional for standard)	6	Yes	No	FSI FW	eMMC
xusb-fw (only for standard)	0.25	Yes	No	XUSB FW	eMMC
pvit	0.25	Yes	No	Partitions Version Info Table	eMMC
pva-fw	2.5	Yes	No	PVA FW	eMMC
kernel (HV image)	10	Yes	No	HV kernel + server VMs + PCT	eMMC
gos0-debug_overlay (For safety prod_debug*)	128	Yes	No	Debug overly for safety images only	eMMC
qnx-gos0 (IFS/ DT for GOSO L3PT)	30	Yes	Yes	GOSO QNX primary IFS image	eMMC
gos0-ifs2(GOSO Secondary IFS)	250	Yes	Yes	GOSO QNX secondary IFS image	eMMC
gos0-ifs2_1(GOSO 2nd Secondary IFS)	250	Yes	Yes	GOSO QNX 2nd secondary IFS image	eMMC
usr-fs (GOSO Root-FS)	2030	Yes	Yes	GOSO root file system	eMMC
gos0-av-rootfs (GOSO AV Root-FS)	12288	Yes	No	Automotive applications rootfs	eMMC

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
gos0-p-sw (NDAS partner Root-FS)	2048	Yes	No	Rootfs for partner	eMMC
gos0-av-factory	64	Yes	No	Factory partition	eMMC
gos1-debug_overlay (For safety prod_debug*) (@)	128	Yes	No	Debug overly for safety images only	eMMC
qnx-gos1 (IFS/DT for GOS1 L3PT) (@)	30	Yes	Yes	GOS1 QNX primary IFS image	eMMC
gos1-ifs2 (GOS1 Secondary IFS) (@)	98	Yes	Yes	GOS1 QNX secondary IFS image	eMMC
gos1-av(GOS1 AV-FS) (@)	256	Yes	Yes	Guest OS1 av file system	eMMC
qnx-update (IFS/DT of DRIVE Update VM L3PT)	24	Yes	No	Update VM QNX primary IFS image and DT	eMMC
qnx-update-fs	128	Yes	No	Filesystem of Update VM QNX	eMMC
gos0-usr-data-ufs (UserData UFS) (#)	8192	Yes	Yes	GOS0 user data	UFS



Note: (@) is a partition only for dual-qnx PCT. (#) is a partition only for HIGH storage configuration option using UFS.

The eMMC/UFS storage chain A/B for Linux PCT NDAS use case:

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
ist-testimg (For safety, Optional for standard)	1280	Yes	No	IST test image	eMMC
ist-runtimeinfo (For safety, Optional for standard)	0.25	Yes	No	IST runtime information	eMMC
ist-resultdata (For safety, Optional for standard)	200	Yes	No	IST result data	eMMC
gr-ist (For safety, Optional for standard)	0.25	Yes	No	gr blob support	eMMC
gos0-crashlogs	1	No	Yes	To store Oops logs	eMMC
secure-os	4	Yes	No	TOS	eMMC
adsp-fw	2	Yes	No	For Audio	eMMC
fsi-fw	6	No	No	FSI FW	eMMC
xusb-fw	0.25	Yes	No	XUSB FW	eMMC
dce-fw	9	Yes	No	DCE FW	eMMC
pvit	0.25	Yes	No	Partitions Version Info Table	eMMC
pva-fw	2.5	Yes	No	PVA FW	eMMC
kernel (HV image)	10	Yes	No	HV kernel + server VMs + PCT	eMMC
guest-linux (Linux GOS VM 3LPT)	3456	Yes	Yes	Linux kernel/ initramfs/DT/ GPT/rootfs	eMMC

Partition Name	Size (in MBytes)	Mandatory	Customizable	Purpose	Partition in
gos0-rw-overlay	256	Yes	Yes	R/W partition for Read-only Rootfs	eMMC
gos0-av-rootfs (GOSO AV Root-FS)	12288	Yes	No	Automotive applications rootfs	eMMC
gos0-p-sw (NDAS partner Root-FS)	2048	Yes	No	Rootfs for partner	eMMC
gos0-av-factory	64	Yes	No	Factory partition	eMMC
qnx-update (IFS/DT of DRIVE Update VM L3PT)	24	Yes	No	Update VM QNX primary IFS image and DT	eMMC
qnx-update-fs	128	Yes	No	Filesystem of Update VM QNX	eMMC
gos0-usr-data-ufs (UserData UFS) (#)	8192	Yes	Yes	GOSO user data	UFS

**Note:**

- > (#) is a partition only for HIGH storage configuration.
- > For mandatory partition, only size of partition is customizable. For non-mandatory partition, all partition-attributes are customizable or it can be removed/split.

NV Pre-Allocated Stream IDs

The following table describes the NV pre-allocated stream IDs.

Partition Name	Linux	Linux Android	QNX	QNX: DRIVE Update	UFS ID	Linux UFS ID	QNX Static Memory Mapping Start	Size
A_ufs_chain				TEGRA_SID_TEGRA_SID_NISO0_0x94000000x01000000				
B_ufs_chain				TEGRA_SID_TEGRA_SID_NISO0_0x95000000x01000000				

Partition Name	Linux	Linux Android	QNX	QNX: DRIVE Update	UFS ID	Linux UFS ID	QNX Static Memory Mapping Start	Size
qnx-update-fs				TEGRA_SID_TEGRA_SID_NISO0_0x96000000x01000000				
pers-ota				TEGRA_SID_TEGRA_SID_NISO0_0x97000000x01000000				
gos0-m-cache-ufs	Y		Y	TEGRA_SID_TEGRA_SID_NISO0_0xd0000000x01000000				
gos0-edr-ufs	Y		Y	TEGRA_SID_TEGRA_SID_NISO0_0xd9000000x01000000				
gos0-sar-ufs	Y		Y	TEGRA_SID_TEGRA_SID_NISO0_0xd8000000x01000000				
gos0-shared-pers	Y		Y	TEGRA_SID_TEGRA_SID_NISO0_0xda000000x01000000				
gos0-ufs	Y		Y	TEGRA_SID_TEGRA_SID_NISO0_0xdb000000x01000000				
gos0-demo-ufs	Y		Y	TEGRA_SID_TEGRA_SID_NISO0_0xdc000000x01000000				
gos0-usr-data-ufs	Y		Y	TEGRA_SID_TEGRA_SID_NISO0_0xd1000000x01000000				
gos0-ifs2			Y	TEGRA_SID_NISO0_0xd3000000x01000000				
gos0-ifs2_1			Y	TEGRA_SID_NISO0_0xd4000000x01000000				
usr-fs			Y	TEGRA_SID_NISO0_0xd5000000x01000000				
gos0-efs2			Y	TEGRA_SID_NISO0_0xd6000000x01000000				
gos0-efs3			Y	TEGRA_SID_NISO0_0xd7000000x01000000				
debug_overla			Y	TEGRA_SID_NISO0_0xdc000000x01000000				

Partition Name	Linux	Linux Android	QNX	QNX: DRIVE Update	UFS ID	Linux UFS ID	QNX Static Memory Mapping Start	Size
guest-linux	Y				TEGRA_SID_NIS00			
gos0-rw-overlay	Y				TEGRA_SID_NIS00			
UDA		Y			TEGRA_SID_NIS00			
GOS1		Y			TEGRA_SID_NIS00			
GOS0		Y			TEGRA_SID_NIS00			

Chapter 6. Embedded Software Components

The following sections describe the embedded software components used to build and deploy applications for DRIVE AGX hardware acceleration engines.

6.1 Graphics Programming

Use the information in this section to understand graphics programming for this release. This information includes topics such OpenGL ES programming tips, and recommendations for managing binary shader programs.

For information on the graphics specifications and extensions supported in this release, see:

- > [OpenGL ES 2/3 Specs and Extensions in the API Reference](#)
- > [EGL and EGL Extensions in the API Reference](#)
- > [Vulkan Specs and Extensions in the API Reference](#)
- > [Vulkan SC Specs and Extensions in the API Reference](#)

6.1.1 OpenGL ES Programming Tips



Note:

NVIDIA does not recommend using OpenGL in a safety environment.

This topic is for readers who have some experience programming OpenGL ES and want to improve the performance of their OpenGL ES application. It aims at providing recommendations on getting the most out of the API and hardware resources without diving into too many architectural details.

6.1.1.1 Programming Efficiently

Some of the recommendations in this topic are incompatible with each other. One must consider the trade-offs between CPU load, memory, bandwidth, shader processing power, precision, and image quality. Premature assumptions about the effectiveness of trade-offs should be avoided. The only definite answers come from benchmarking and looking at rendered images!

The items below are not ordered according to importance or potential performance increase. The identifiers in parentheses exist only for reference.

6.1.1.1.1 State

Inefficient management of GL state leads to increased CPU load that may limit the amount of useful work the CPU could be doing elsewhere. Reducing the number of times rendering is paused due to GL state change will increase the chance of realizing the potential throughput of the GPU. The main point in this section is: do not modify or query GL state unless absolutely necessarily.

Do not set any state redundantly (S1)

All relevant GL state should be initialized during application initialization and not in the main render loop. For instance, occasionally `glClearDepthf`, `glColorMask`, or `glViewport` finds its way into the application render loop even though the values passed to these functions are always constant. Other times they are set unconditionally in the loop, just in case their values have changed per frame. Only call these functions when the values actually do need to change. Additionally, do not automatically set state back to some predefined value (e.g., the GL defaults). That idiom might be useful while developing your application as it makes it easier to re-order pieces of rendering code, but it should not be done in production code without a very good reason.

Avoid querying any GL state in the render loop (S2)

When a GL context is created, the state is initially well-defined. Every state variable has a default value that is described in the OpenGL ES specification ("State Tables"). Except when compiling shaders, determining available extensions, or the application needs to query implementation specific constants, there should be no need to query any GL state. These queries can almost always be done in initialization. Well-written applications check for GL errors in debug builds. If no errors are reported as a result of changing state, it is assumed that the changes are now part of the new GL state. For these two reasons, the current state is always known and you should almost never need to query any GL state in a loop. If an application frequently calls functions that begin with `glIs*` or `glGet*`, these calls should be tracked down and eliminated.

Batch on shared state (S3)

An efficient approach to reduce the number of state changes is batching together all draw calls that use the same state (shaders, blending, textures, etc.). For instance, not batching on the shader changes has the form:

```
[ UseProgram(21), DrawX1, UseProgram(59), DrawY1,
  UseProgram(21), DrawX2, UseProgram(59), DrawY2 ]
```

Batching on the shaders leads to an improvement (fewer shader changes):

```
[ UseProgram(21), DrawX1, DrawX2,
  UseProgram(59), DrawY1, DrawY2 ]
```

It is quite effective to group draw calls based on the shader programs they use. Reprogramming the GPU by switching between shaders is a relatively costly operation. Within each batch, a further optimization can be made by grouping draw calls that use

the same uniforms, texture objects and other state. Generating a log of all function calls into the OpenGL ES API is a good approach for revealing poor batching. A tool such as PerfHUES can conveniently generate this log without rebuilding the GL application; no change to the source code is necessary.

Do not repeat per object state when binding (S4)

Recall that some state is bound to the object it affects. As that state is stored in the object, you do not need to repeat it when you rebind the object. A very common mistake is setting the texture parameters for filtering and wrapping every time a texture object is bound. Another common mistake is updating uniform variables that have not changed value since the last time the particular shader program was used. In particular, when batching opportunities are limited, repeating per object state generates enormously inefficient GL code that can easily have a measurable impact on framerate.

Enable backface culling whenever possible (S5)

Always enable face culling whenever the back-faces are not visible. Rendering the back-faces of primitives is often not necessary.



Note:

The default GL state has backface culling disabled, so this is one state that should almost always be set during application initialization and be left enabled for the application lifetime.

6.1.1.1.2 Geometry

The amount of geometry, as well as the way it is transferred to the GL, can have a very large impact on both CPU and GPU load. If the application sends geometry inefficiently or too frequently, that alone can create a bottleneck on the CPU side that does not give the GPU enough data to work efficiently. Geometry must be submitted in sizable chunks to realize the potential of the GPU. At the same time, geometry should be minimally defined, stored on the server side, and it should be accessed in a way to get the most out of the two GPU caches that exist before and after vertices are transformed.

Use indexed primitives (G1)

The vertex processing engine contains a cache where previously transformed vertices are stored. It is called Post-TnL vertex cache. Taking full advantage of this cache can lead to very large performance improvement when vertex processing is the bottleneck. To fully utilize it, it is necessary for the GPU to be able to recognize previously transformed vertices. This can only be accomplished by specifying indexed primitives for the geometry. However, for any non-trivial geometry the optimal order of indices will not be obvious to the programmer. If some geometry is complex, and the application bottleneck is vertex processing, then look into computing a vertex order that maximizes the hit ratio of the Post TnL cache. The topic has been thoroughly studied for years and even simple

greedy algorithms can provide a substantial performance boost. Good results have been reported with the algorithm described at the below locations.

Document	URL to Latest
<i>Linear-Speed Vertex Cache Optimisation</i> , by Tom Forsyth, RAD Game Tools (28th September 2006)	URL

There is a free implementation of the algorithm in a library called `vcacne`.

 Note:	<p>The number of vertex attributes and the size of each attribute may determine the efficiency of this cache—it has storage for a fixed number of bytes or active attributes, not a fixed number of vertices. A lot of attribute data per vertex increases the risk of cache misses, resulting in potentially redundant transformations of the same vertices.</p>
--	---

Reduce vertex attribute size and components (G2)

It is important to use an appropriate attribute size and minimize the number of components to avoid wasting memory bandwidth and to increase the efficiency of the cache that stores pre-transformed vertices. This cache is called Pre-TnL vertex cache. For instance, you rarely need to specify attributes in 32 bit FLOATs. It might be possible to define the object-space geometry using 3 BYTES per vertex for a simple object, or 3 SHORTs for a more complex or larger object. If the geometry requires floating-point representation, half-floats (available in extension `OES_vertex_half_float.txt`) may be sufficient. Per vertex colors are accurately stored with 3 x BYTES with a flag to normalize in `VertexAttribPointer`. Texture coordinates can sometimes be represented with BYTES or SHORTs with a flag to normalize (if not tiling).

 Note:	<p>The exception case that normalizing texture coordinates is not necessary if they are only used to sample a cube map texture.</p>
--	---

Vertex normals can often be represented with 3 SHORTs (in a few cases, such as for cuboids, even as 3 BYTES) and these should be normalized. Normals can even be represented with 2 components if the direction (sign) of the normal is implicit, given its length is known to be 1. The remaining coordinate can be derived in a vertex shader (e.g. $z = \text{SQRT}(1 - x * x + y * y)$) if memory or bandwidth (rather than vertex processing) is a likely bottleneck.

An optimal OpenGL ES application will take advantage of any characteristics specific to the geometry. For instance, a smooth sphere uses the normalized vertex coordinates as normal—these are trivially computed in a vertex shader. It is important to benchmark intermediate results to ensure the vertex processing engine is not already saturated. Finally remember, if some attribute for a primitive or a number of primitives is constant

for the same draw call, then disable the particular vertex attribute index and set the constant value with `VertexAttrib` instead of replicating the data.

Pack vertex attributes (G3)

Vertex attributes normally have different sets of attributes that are completely unrelated. Unlike uniform and varying variables in shader programs, vertex attributes do not get automatically packed, and the number of vertex attributes is a limited resource. Failure to pack these attributes together may lead to limitations sooner than expected. It is more efficient to pack the components into fewer attributes even though they may not be logically related. For instance, if each vertex comes with two sets of texture coordinates for multi-texturing, these can often be combined these into one attribute with four components instead of two attributes with two components. Unpacking and swizzling components is rarely a performance consideration.

Choose an appropriate vertex attribute layout (G4)

There are two commonly used ways of storing vertex attributes:

- > Array of structures
- > Structures of arrays

An **array of structures** stores the attributes for a given vertex sequentially with an appropriate offset for each attribute and a non-zero stride. The stride is computed from the number of attribute components and their sizes. An array of structures is the preferred way of storing vertex attributes due to more efficient memory access. If the vertex attributes are constant (not updated in the render loop) there is no question that an array of structures is the preferred layout.

In contrast, a **structure of arrays** stores the vertex attributes in separate buffers using the same offset for each attribute and a stride of zero. This layout forces the GPU to jump around and fetch from different memory locations as it assembles the needed attributes for each vertex. The structure of arrays layout is therefore less efficient than an array of structures in most cases. The only time to consider a structure of arrays layout is if one or more attributes must be updated dynamically. Strided writes in array of structures can be expensive relative to the number of bytes modified. In this scenario, the recommendation is to partition the attributes such that constant and dynamic attributes can be read and written sequentially, respectively. The attributes that remain constant should be stored in an array of structures. The attributes that are updated dynamically should be stored in smaller separate buffer objects (or perhaps just a single buffer if the attributes are updated with the same frequency).

Use consistent winding (G5)

The geometry winding (clockwise or counter-clockwise) should be determined up front and defined in code. The geometry face that is culled by GL can be changed with the `FrontFace` function, but having to switch back and forth between winding for different geometry batches during rendering is not optimal for performance and can be avoided in most cases.

Always use vertex and index buffer objects (G6)

Recall that vertices for geometry can either be sourced from application memory every time it is rendered or from buffers in graphics memory where it has been stored previously. The same applies to vertex array indices. To achieve good performance, you should never continuously source the data from application memory with `DrawArrays`. Buffer objects should always be used to store both geometry and indices. Check that no code is calling `DrawArrays`, and that no code is calling `DrawElements` without a buffer bind.

The default buffer usage flag when allocating buffer objects is `STATIC_DRAW`. In many cases this will lead to fastest access.



Note:

`STATIC_DRAW` does not mean one can never write to the buffer (although any writing to a buffer should always be avoided as much as possible). A `STATIC_DRAW` flag may in fact be the appropriate usage flags, even if the buffer contents are updated every few frames. Only after careful benchmarking and arriving at conclusive results should changing the usage flag to one of the alternatives (`DYNAMIC_DRAW` or `STREAM_DRAW`) be considered.

Batch geometry into fewer buffers and draw calls (G7)

There are only so many draw calls, or batches of geometry, that can be submitted to GL before the application becomes CPU bound. Each draw call has an overhead that is more or less fixed. Therefore, it is very important to increase the sizes of batches whenever possible. There does not need to be a one-to-one correspondence between a draw call and a buffer—a large vertex buffer can store geometry with a similar layout for multiple models. One or more index buffers can be used to select the subset of vertices needed from the vertex buffer. A common mistake is to have too many small buffers, leading to too many draw calls and thus high CPU load. If the number of draw calls issued in any given frame goes into many hundreds or thousands, then it is time to consider combining similar geometry in fewer buffers and use appropriate offsets when defining the attribute data and creating the index buffer.

Unconnected geometry can be stitched together with degenerate triangles (alternatively, by using extension `NV_primitive_restart2` when available). Degenerate triangles are triangles where two or more vertices are coincident leading to a null surface. These are trivially rejected and ignored by the GPU. The benefit from stitching together geometry with degenerate triangles, such that fewer and larger buffers are needed, tends to outweigh the minor overhead of sending degenerates triangles down the pipeline. If geometry batches are being broken up to bind different textures, then look at combining several images into fewer textures (T5).

Use the smallest possible data type for indices (G8)

When the geometry uses relatively few vertices, an index buffer should specify vertices using only `UNSIGNED_BYTE` instead of `UNSIGNED_SHORT` (or an even larger integer type if the ES2 implementation supports it). Count the number of unique vertices per buffer and choose the right data type. When batching geometry for several unrelated models

into fewer buffer objects (G7), then a larger data type for the indices may be required. This is not a concern compared to the larger performance benefits of batching.

Avoid allocating new buffers in the rendering loop (G9)

If the application frequently updates the geometry, then allocate a set of sufficiently large buffers when the application initializes. A `BufferData` call with a `NULL` data pointer will reserve the amount of memory you specify. This eliminates the time spent waiting for an allocation to complete in the rendering loop. Reusing pre-allocated buffers also helps to reduce memory fragmentation.



Note:

Writing to a buffer object that is being used by the GPU can introduce bubbles in the pipeline where no useful work is being done. To avoid reducing throughput when updating buffers, consider cycling between multiple buffers to minimize the possibility of updating the buffer from which content is currently being rendered.

Cull early and often (G10)

The GPU will not rasterize primitives when all of its vertices fall outside the viewport. It also avoids processing hidden fragments when the depth test or stencil test fails (P4). However, this does not mean that the GPU should do all the work in deciding what is visible. In the case of vertices, they need to be loaded, assembled and processed in the vertex shader, before the GPU can decide whether to cull or clip parts of the geometry. Testing a single, simple bounding volume that encloses the geometry against the current view frustum on the CPU side is a lot faster than testing hundreds of thousands of vertices on the GPU. If an application is limited by vertex processing, this is definitely the place to begin optimizing. Spheres are the most efficient volumes to test against and the volume of choice if geometry is rotational symmetrical. For some geometry, spheres tend to lead to overly conservative visibility acceptance. A rectangular cuboid (box) is only a slightly more expensive test but can be made to fit more tightly on most geometry. Hierarchical culling can often be employed to reduce the number of tests necessary on the CPU. Efficient and advanced culling algorithms have been heavily researched and published for many years. You can find a good introduction in the survey at the below locations.

6.1.1.3 Shader Programs

Writing efficient shaders is critical to achieving good performance. One should treat shaders like pieces of code that run in the inner-most loops on a CPU. There is a very high cost to littering these with conditionals or recomputing loop invariants. Before optimizing an expensive vertex shader, make sure geometry that is entirely outside the view frustum is being culled on the CPU. Before optimizing an expensive fragment

shader, make sure the application is not generating an excess number of fragments with it.



Note:

When optimizing shaders, any source code that does not contribute to its output variables is optimized out by the compiler. This feature can be exploited to gain knowledge about whether the shader is part of the current bottleneck by multiplying the output variable with a null vector to reduce the workload and then measure if frame rate improves. Conversely, at the final stages of optimization one can quickly measure if there is headroom for increasing workload to offload computations to shader unit or to improve image quality by adding meaningless but expensive ALU instructions, or texture sampling, to the output variables.

Move computations up the pipeline (P1)

As the rendering pipeline is traversed from the CPU to the vertex processor and then to the fragment processor, the required workload tends to increase a few orders of magnitude each time. Computations constant per model, per primitive or per vertex do not belong in the fragment processor and should be moved up to the vertex processor or earlier. Per draw call computations do not belong in the vertex processor and should be moved to the CPU. For instance, if lighting is done in eye-space, the light vector should be transformed into eye-space and stored in a uniform rather than repeating this for each vertex or, even worse, per fragment. The light vector should naturally be stored pre-normalized. Usually the light vector computations are constant for the draw call, so they do not belong in any shader.

Do not write large or generalized shaders (P2)

It is critical to resist the temptation to write shader programs that take different code paths depending on whether one or more constant variable have a particular value. Uniforms are intended as constants for one (or hopefully many) primitives—they are not substitutes for calling `UseProgram`. Shaders should be minimal and specialized to the task they perform. It is much better to have many small shaders that run fast than a few large shaders that all run slow. Code re-use (when source shaders are supported) should be handled at the application level using the `ShaderSource` function. If the advice here of not writing generalized shaders goes against the conflicting goal of minimizing shader and state changes, smaller and more specialized shaders are generally preferred. Additionally, be careful with writing shader functions intended for concatenation into the final shader source code - shared functions tend to be overly generic and make it harder to exploit possible shortcuts.

Take advantage of application specific knowledge (P3)

Application specific knowledge can be used to simplify or avoid computations. Math shortcuts should be pursued everywhere because there are optimizations that the shader compiler or the GPU cannot make. For instance, rendering screen-aligned primitives is common for 2D user interface and post-processing effects. In this case, the entire `modelview` transformation is avoided by defining the vertices in NDC (normalized device coordinates). A full-screen quad has its vertex coordinates in the [-1.0,1.0] range

so these can be passed directly from the vertex attribute to `gl_Position`. The types of matrix transformations applied in the application when creating the `modelview` matrix should be tracked and exploited when possible. For instance, an orthonormal matrix (e.g. no non-uniform scaling) leads to an opportunity to avoid computations when transforming normals with the inverse-transpose sub-matrix.

Optimize for depth and stencil culling (P4)

The GPU can quickly reject fragments based on depth or stencil testing before the fragment shader is executed. The depth complexity of a scene is the number of times each fragment gets written. Depth complexity can be measured by incrementing values in a stencil buffer. A high depth complexity in a 3D scene can be a result of rendering opaque objects in a non-optimal order. The worst case is rendering back-to-front (aka painter's algorithm) because it leads to a large number of fragments being overdrawn. An application with high depth complexity should ensure that opaque objects are rendered sorted front-to-back order with depth testing enabled. Straightforward rendering of 2D user interfaces also leads to a high depth complexity that can often be decreased with the same technique but also by using the stencil buffer to mask fragments. Applications that are heavily fragment limited can be sped up significantly with clever use of these techniques—sometimes up to a factor of 10 or more.

If vertex processing is not a bottleneck, it is worthwhile to run experiments that prime the depth buffer in a first pass. Disable all color writes with `ColorMask` on the first pass. The fragments in the depth buffer can then serve as occluders in a second pass when color writes are enabled and the expensive fragment shaders are executed. Disable depth writes with `DepthMask` in the second pass since there is no point in writing it twice.

Do not discard fragments, or modify depth, unless absolutely necessary (P5)

Some operations prevent the hardware from enabling its automatic optimization that rejects fragments early in the pipeline (early-Z). In particular, the `discard` operation that discards fragments based on some criteria will disable early-Z on some platforms. It is critical to limit the use of discarding as much as possible (e.g., alpha testing)—unless depth writing can be disabled. Another example is found in the `GL_NV_fragdepth` extension available on some platforms, where the depth value can be written from the fragment shader. This operation also forces the GPU to opt out of Early-Z reject in order to ensure correct rendering.

Avoid conditionals in shaders when possible (P6)

Fragments are processed in chunks and both branches of a conditional may need to be evaluated before the result of the false branch can be discarded by the GPU. Be careful with assuming that conditionals skip computations and reduce the workload. This warning is particularly relevant to fragment shaders. Benchmarking shaders can determine if conditionals in the vertex or fragment shaders actually end up decreasing the workload. Some conditional code can be rewritten in terms of algebra and/or built-in functions. For instance, the dot product between a normal and a light vector may be negative in which case the result is not needed in a lighting equation. Instead of:

```
if (nDotL > 0.0) ...
```

the value can be clamped with:

```
clamp(nDotL, 0.0, 1.0)
```

and unconditionally used in the result (the negative value results in a zero-product). Clamp may be faster than max and/or min for the 0.0 and 1.0 cases, but as always benchmarking will have the final say in the matter. Another reason to make an effort of avoiding conditionals in fragment shaders is that mipmapped textures return undefined results when executed in a block statement that is conditional on run-time values. Although the GLSL functions `texture*Lod` can be used to bias or specify the mipmap LOD, it is expensive to manually derive the mipmap LOD. In addition, these LOD biasing samplers may not run as fast as the non-LOD samplers.

Use appropriate precision qualifiers (P7)

Recall that the default precision in vertex shaders is `highp`, and that fragment shaders have no default precision until explicitly set. Precision qualifiers can be valuable hints to the compiler to reduce register pressure and may improve performance for several reasons. Low precision may run twice as fast in hardware as high precision. These optimizations can be approached by initially using `highp` and gradually reducing the precision to `lowp` until rendering artifacts appear; if it looks good enough, then it is good enough. As a rule of thumb, vertex position, exponential and trigonometry functions need `highp`. Texture coordinates may need anything from `lowp` to `highp` depending on texture width and height. Many application-defined uniform variables, interpolated colors, normals and texture samples can usually be represented using `lowp`. However, floating-point texture samplers need more than low precision - this is one of several reasons to minimize the use of floating-point textures (T6).

Use the built-in functions and variable (P8)

The built-ins have a higher chance of compiling optimally and may even be implemented in hardware. For instance, do not write shader code to determine the forward primitive face or compute the reflection vector in terms of dot-products and algebra; instead, use the built-in variable `gl_FrontFacing` or the built-in function `reflect`, respectively.

Consider encoding complex functions in textures (P9)

Shaders normally contain both arithmetic (ALU) and texture operations. A batch of ALU operations may hide the latency of fetching samples from texture because they occur in parallel. If a shader is the primary bottleneck, and when the ALU operations significantly outnumber the texture operations, it is worthwhile to investigate if some of these operations can be encoded in textures. Sub-expressions in shaders can sometimes be stored in LUTs (look-up-tables). LUTs can sometimes be implemented in textures with sufficient precision and accessed as 1D or 2D textures with NEAREST filtering.



Note:

The old trick of using cubemaps to normalize vectors is most likely a performance loss on discrete GPUs. If you pursue this idea, then make sure to benchmark to determine if you have improved or worsened the performance!

Limit the amount of indirect texturing (P10)

Indirect texturing can sometimes be useful, but when the result of a texture operation depends on another texture operation, the latency of texture sampling is difficult to hide. It also tends to lead to scattered reads that minimize the benefit of the texture cache. Indirect texturing can sometimes be reduced, or avoided, at the expense of memory. Whether that trade-off makes sense should of course be analyzed and benchmarked.

Do not let GLSL syntax obscure math optimizations (P11)

The GLSL shading language conveniently overloads arithmetic operators for vectors and matrices. Care must be taken to not miss optimization opportunities due to this syntax simplification. For instance, a rotation matrix can, but should not, be defined as homogenous 4x4 matrix just because the other operand is a `vec4` vector. A generalized rotation matrix should be described only as a 3 x 3 matrix and applied to `vec3` vectors. And a rotation around basic vectors can be done even more efficiently than `mat3 * vec3` by directly accessing the relevant vector and matrix components with `cos` and `sin`. Take advantage of any specific application knowledge to reduce the number of needed scalar instructions.

Only normalize vectors when it is necessary (P12)

Normalization of vectors is required to efficiently calculate the angle between them, or perhaps more typically, the diffuse component in many commonly used lighting equations. In theory, normalization is required for geometry normals after having transformed them with the normal matrix. In practice, it may not matter depending on the composition of the matrix. It is a common mistake to normalize vectors where it is not necessary, or at least not visually discernible. As a result, the application may run slower for no good reason. For instance, consider the case of interpolating vertex normals in order to compute lighting per fragment (i.e., Phong shading). If the normal matrix only rotates, there is little reason to normalize the normal vectors before interpolating.



Note:

Barycentric interpolation will not preserve the unit length of these vectors. So normals that are interpolated in varying variables do must be normalized to ensure the dot-product with the light vector obeys the cosine emission law.

6.1.1.1.4 Textures

Textures consume the largest amount of the available memory and bandwidth in many applications. One of the best places to look for improvements when short on memory or bandwidth to optimize texture size, format and usage. Careless use of texturing can degrade the frame rate and can even result in inferior image quality.

Use texture compression whenever possible (T1)

Texture compression brings several benefits that result from textures taking up less memory: compressed textures use less of the available memory bandwidth, reduces download time, and increases the efficiency of the texture cache. The

`texture_compression_s3tc` and `texture_compression_latc` extensions both provide block-based lossy texture compression that can be decompressed efficiently in hardware. The S3TC extension gives 8:1 or 4:1 compression ratio and is suitable for color images with 3 or 4 channels (with or without alpha) with relatively low-frequency data. Photographs and other images that compress satisfactory with JPEG are great candidates for S3TC. Images with hard and crisp edges are less good candidates for S3TC and may appear slightly blurred and noisy. The LATC extension yields a 2:1 compression ratio, but improves on the quality and can be useful for high resolution normal maps. The third coordinate is derived in the fragment shader—be sure to benchmark if the application can afford this trade-off between memory and computations! Unlike S3TC, the channels in LATC are compressed separately, and quantization is less hard. Using texture compression does not always result in lower perceived image quality, and with these extensions one can experiment with increasing the texture resolution for the same memory. There are off-line tools to compress textures (even if a GL extension supports compressing them on-the-fly). Search the NVIDIA developer website for "Texture Tools".

Use mipmaps when appropriate (T2)

Mipmaps should be used when there is not an obvious one-to-one mapping between texels and framebuffer pixels. If texture minification occurs in a scene and there are no mipmaps to access, texture cache utilization will be poor due to the sparse sampling. If texture minification occurs more often than not, then the texture size may be too large to begin with. Coloring the mipmap levels differently can provide a visual clue to the amount of minification that is occurring. When reducing the texture size, it may also be worthwhile to perform experiments to see if some degree of magnification is visually acceptable and if it improves frame rate.

Although the `GenerateMipmap` function is convenient, it should not be the only option for generating a mipmap chain. This function emphasizes execution speed over image quality by using a simple box filter. Generating mipmaps off-line using more advanced filters (e.g. Lanczos/Sinc) will often yield improved image quality at no extra cost. However, `GenerateMipmap` may be preferable when generating textures dynamically due to speed. One of the only situations where you do not want to use mipmaps is if there is always a one-to-one mapping between texels and pixels. This is sometimes the case in 3D, but more often the case for 2D user interfaces. Recall that a mipmapped texture takes up 33% more storage than un-mipmapped, but they can provide much better performance and even better image quality through reduced aliasing.

Use the smallest possible textures size (T3)

Always use the smallest possible texture size for any content that gives acceptable image quality. The appropriate size of a texture should be determined by the size of the framebuffer and the way the textured geometry is projected onto it. But even when you have a one-to-one mapping between texels and framebuffer pixels, there may be cases where a smaller size can be used. For instance, when blending a texture on the existing content in the entire framebuffer, the texture does not necessarily have to be the same width and height as the framebuffer. It may be the case that a significantly smaller texture that is magnified will produce results that are good enough. The bandwidth that is saved from using a smaller and more appropriately sized texture can instead be spent where it actually contributes to better image quality or performance.

Use the smallest possible texture format and data type (T4)

If hardware accelerated texture compression cannot be used for some textures, then consider using a texture format with fewer components and/or fewer bits per component. Textures for user interface elements sometimes have hard edges or color gradients that result in inferior image quality when compressed. The S3TC algorithm make assumptions that changes are smooth and colors values can be quantized. If these assumptions do not fit a particular image, but the number of unique colors is still low, then experiment with storing these in a packed texture format using 16 bit/texel (e.g. UNSIGNED_SHORT_5_6_5). Although the colors are remapped with less accuracy it may not be noticeable in the final application. Grayscale images should be stored as LUMINANCE and tinted images can sometimes be stored the same way with the added cost of a dot product with the tint color. If normal maps do not compress satisfactory with the LATC format, then it may be possible to store two of the normals coordinates in uncompressed LUMINANCE_ALPHA and derive the third in a shader assuming the direction (sign) of the normal is implicit (as is the case of a heightmap terrain).



Note:

When optimizing uncompressed textures, the exception case that 24-bit (RGB) textures are not necessarily faster to download or smaller in memory than 32-bit (RGBA) on most GPUs. In this case, it may be possible to use the last component for something useful. For instance, if there already is an 8-bit greyscale texture that is needed at the same time as an opaque color texture, that single component texture can be stored in the unused alpha component of a 32-bit (RGBA). The component could define a specular/reflectance map that describe where and to what degree light is reflected. This is useful for terrain satellite imagery or land cover textures with water/snow/ice areas or for car textures with their metal and glass surfaces or for textures for buildings with glass windows.

Store multiple images in each texture object (T5)

There is no requirement that there is a one-to-one mapping between an image and a texture object. Textures objects can contain multiple distinct images. These are sometimes referred to as a "texture atlas" or a "texture page". The geometry defines texture coordinates that only reference a subset of the texture. Texture atlases are useful for minimizing state changes and enables larger batches when rendering. For example, residential houses and office buildings and factories might all use distinct texture images. But the geometry and vertex layout for each is most likely identical so these could share the same buffer object. If the distinct images are stored in a texture atlas instead of as separate textures, then these different kinds of buildings can all be

rendered more efficiently in the same draw call (G7). The texture object could be a 2D texture, a cubemap texture or an array texture.


Note:

Note that if mipmaping is enabled, the sub-textures in an atlas must have a border wide enough to ensure that smaller mipmaps are not generated using texels from neighboring images. And if texture tiling (REPEAT or MIRRORRED_REPEAT) is needed for a sub-image then it may be better to store it outside the texture atlas.

Emulating either wrapping mode in a shader by manipulating texture coordinates is possible, but not free. A cubemap texture can sometimes be useful since wrapping and filtering apply per face, but the texture coordinates used must be remapped to a vec3 which may be inconvenient. If all the sub-images have the same or similar size, format and type (e.g. image icons), the images are a good candidate for the array texture extension if supported. Array textures may be more appropriate here than a 2D texture atlas where mipmaping and wrapping restrictions have to be taken into consideration.

Float textures are always expensive (T6)

Textures with a floating-point format should be avoided whenever possible. If these textures are simply being used to represent a larger range of values, it may be possible to replace these with fixed point textures and scaling instructions. For instance, unsigned 16-bit integers cannot even accurately be represented by half-precision floats (FP16). These would have to be stored using single precision (FP32) leading to twice the memory and bandwidth requirements. It might be better to store these values in two components using 8 bits (LA8) and spend ALU instructions to unpack them in a shader.


Note:

Floating-point textures may not support anything better than nearest filtering.

Prefer power-of-two (POT) textures in most cases (T7)

Although Non-Power-of-Two (NPOT) textures are supported in ES2 they come with a CLAMP_TO_EDGE restriction on the wrapping mode (unless relaxed by an extension). More importantly, they cannot be mipmapped (unless relaxed by an extension). For that reason, POT textures should be used when there is not significant memory and bandwidth to be saved from using NPOT. However, an NPOT texture may be padded internally to accommodate alignment restrictions in hardware and that the amount of memory saved might not be quite as large as the width and height suggests. As a rule of thumb, only large (i.e., hundreds of texels) NPOT textures will effectively save a significant amount of memory over POT textures.

Update textures sparingly (T8)

Writing to GPU resources can be expensive—it applies to textures as well. If texture updates are required, then determine if they really need to be updated per frame or if the same texture can be reused for several frames. For environment maps, unless the lighting or the objects in the environment have been transformed (e.g., moved/rotated) sufficiently to invalidate the previous map, the visual difference may not be noticeable

but the performance improvement can be. The same applies to the depth texture(s) used for shadow mapping algorithms.

Update textures efficiently (T9)

When updating an existing texture, use `TexSubImage` instead of re-defining its entire contents with `TexImage` when possible.



Note:

When using `TexSubImage` it is important to specify the same texture format and data type with which that the texture object was defined; otherwise, there may be an expensive conversion as texels are being updated.

If the application is only updating from a sub-rectangle of pixels in client memory, then remember that the driver has no knowledge about the stride of pixels in your image. When the width of the image rectangle differs from the texture width, this normally requires a loop through single pixel rows calling `TexSubImage` repeatedly while updating the client memory offsets with pointer arithmetic. In this case, the `unpack_subimage` extension can be used (if supported) to set the `UNPACK_ROW_LENGTH` pixelstore parameter to update the entire region with one `TexSubImage` call.

Partition rendering based on alpha blending/testing (T10)

It is sometimes possible to improve performance by splitting up the rendering based on whether alpha blending or alpha testing is required. Use separate draw calls for opaque geometry so these can be rendered with maximum efficiency with blending disabled. Perform other draw calls for transparent geometry with alpha blending enabled, taking into account the draw ordering that transparent geometry requires. As always, benchmarks should be run to determine if this improves or reduces the frame rate since you will be batching less by splitting up draw calls.

Filter textures appropriately (T11)

Do not automatically set expensive texture filters and enable anisotropic filtering. Remember that nearest-neighbor filtering always fetches one texel, bilinear filtering fetches up to four texels and trilinear fetches up to eight texels. However, it can be incorrect to draw assumptions about the performance cost based on this. Bilinear filtering may not cost four times as much as nearest filtering, and trilinear can be more or less than twice as expensive as bilinear. Even though textures have mipmaps, it does not automatically mean trilinear filtering should be used. That decision should be made entirely from observing the images from the running application. Only then can a judgment be made if any abrupt changes between mipmap levels are visually disturbing enough to justify the cost of interpolating the mipmaps with trilinear filtering. The same applies to anisotropic filtering, which is significantly more expensive and bandwidth intensive than bilinear or trilinear filtering. If the angle between the textured primitives and the projection plane (e.g. near plane) is never very large, there is nothing to be gained from sampling anisotropically and there is potentially lower performance. Therefore, an application should start off with the simplest possible texture filtering and only enable more expensive filtering after users have inspected the output images. It might be worthwhile to benchmark the changes and take notes along the way. This will provide

a better indication of the relative cost of filtering method and if concessions must be made if the performance budget is exceeded.

Try to exploit texture tiling (T12)

It is common for images to contain the same repeated pattern of pixels. Or an image might repeat a few patterns that are close enough in similarity that they could be replaced with a single pattern without impacting image quality. Tiling textures saves on memory and bandwidth. Some image processing applications can identify repeated patterns and can crop them so they can be tiled infinitely without seams when using textures with the REPEAT wrap mode. Sometimes even a quarter of a tile or shingle may be sufficient to store while using MIRRORED_REPEAT. Consider if tiling variation can be restored or achieved with multi-texturing, using for instance a less expensive grey-scale texture that repeats at a different frequency to modulate the texels from the tiled texture.

Use framebuffer objects (FBO) for dynamically generated textures (T13)

OpenGL ES comes with functions for copying previously rendered pixels from a framebuffer into a texture (`TexCopyImage`, `TexCopySubImage`). These functions should be avoided whenever possible for performance reasons. It is better to bind a framebuffer object with a texture attachment and render directly to the texture. Make sure you check for framebuffer completeness.



Note:

Not all pixel formats are color-renderable. Formats with 3 or 4 components in 16 or 32 bits are color-renderable in OpenGL ES 2.0, but LUMINANCE and/or ALPHA may require a fall-back to `TexCopyImage` functions.

6.1.1.1.5 Miscellaneous

This topic contains miscellaneous OpenGL ES programming tips.

Avoid reading back the framebuffer contents (M1)

Reading back the framebuffer flushes the GL pipeline and limits the amount CPU/GPU parallelism. Reading frequently or in the middle of a frame stalls the GPU and limits the throughput with lower frame rate as a result. If the buffer contents must be read back (perhaps for picking 3D objects in a complex scene), it should be done minimally and scheduled at the beginning of the next frame. In the special case that the application is reading back into a sub-rectangle of pixels in client memory, the `pack_subimage` extension (if supported) is very useful. Setting the `PACK_ROW_LENGTH` pixel store parameter will reduce the loop overhead that will otherwise be necessary (T9).

Avoid clearing buffers needlessly (M2)

If the application always covers the entire color buffer for each frame, then bandwidth can be saved by not clearing it. It is a common mistake to call `Clear(GL_COLOR_BUFFER_BIT)` when it is not necessary. If only part of the color buffer is modified, then constrain pixel operations to that region by enabling scissor testing

and define a minimal scissor box for the region. The same applies to depth and stencil buffers if full screen testing is not needed.

Disable blending when it is not needed (M3)

Most blending operations require a read and a write to the framebuffer.



Note:

Memory bandwidth is often doubled when rendering with blending is enabled. The number of blended fragments should be kept to a minimum—it can drastically speed up the GL application.

Minimize memory fragmentation (M4)

Buffer objects and `glTexImage*` functions are effectively graphics memory allocations. Reusing existing buffer objects and texture objects will reduce memory fragmentation. If geometry or textures are generated dynamically, the application should allocate a minimal pool of objects for this purpose during application initialization. It may be that two buffers or textures used in a round-robin fashion are optimal for reducing the risk that the GPU is waiting on the resource. Also, recall that sampling a texture that is being rendered to, at the same time, is undefined. This can be another reason to alternate between objects. For more information, see [Memory Fragmentation](#) in this appendix.

6.1.1.1.6 Optimizing OpenGL ES Applications

Optimization is an iterative process. It can be time consuming, especially without prior experience determining where bottlenecks tend to occur. Effort should be directed towards the critical areas instead of starting a random place in the rendering code. When the graphics application is complex it may be difficult to know where to start or exactly where optimizations will yield the best return.

Partition the analysis into manageable chunks

Many rendering applications are complex and consist of hundreds of objects. But usually they consist of logically separate rendering code. For example, a rendered image may consist of roads, buildings, landmarks, points of interest, sky, clouds, buildings, water, terrain, icons, and a 2D user interface. It is helpful to write the GL application such that rendering of each type of object can be disabled easily. This allows easy identification of the most expensive objects when benchmarking and therefore makes optimizing the rendering code more manageable.

Become familiar with bottlenecks in the graphics pipeline

It is important to begin optimizations by identifying the performance bottlenecks at the different stages in the graphics pipeline. Because the work introduced in the beginning of the pipeline normally affects the work needed at later stages, it often makes sense to work backwards from the end of the pipeline. An introduction to identifying graphics bottlenecks can be found in the GPU Gems book, "Chapter 28. Graphics Pipeline Performance" (Cem Cebenoyan, NVIDIA).

6.1.1.2 Avoiding Memory Fragmentation

Memory Fragmentation generally is a bad thing. This is especially true for computer graphics applications. In addition to avoiding system memory fragmentation, a graphics application should strive to avoid video memory fragmentation as well.

Fortunately, controlling video memory fragmentation has techniques very similar to those used to avoid system memory fragmentation. Since system memory fragmentation control is fairly well known, this document will only treat system memory issues in passing and focuses on video memory techniques.

6.1.1.2.1 Video Memory Overview

Video memory is much more heterogeneous than system memory.

NVIDIA video memory allocation algorithms have to take the following into account:

- There are multiple types of video memory **types**. The number and names of the types vary by GPU model, but GPUs generally have at least two; linear, which is essentially unformatted, and one or more GPU specific types. The GPU tracks different types of memory, and will access and write them differently. The types are important because GPU native types can be faster for a given set of operations; in some GPU architectures, the difference is small, on the order of 10-15%. On others, it can be quite large, more than 100% faster than linear memory.
- Video memory is often banked, especially for mipmapped textures. In most architecture, alternating mipmap levels for a given texture must be put in separate banks. This separation is mandatory in most NVIDIA GPUs.
- In addition to the restrictions above, different memory regions have different alignment restrictions, to match host pages, improve DMA performance, or speed up framebuffer scan out. These alignment requirements may be orthogonal to the memory types, adding further complication.
- The allocator may have other special restrictions that enhance performance, such as distributing allocations to a sequence of different banks to improve allocation speed.
- These extra constraints complicate the video memory allocator, and make allocations much more sensitive to reductions in available video memory. This is the major reason why NVIDIA does not support multiple independent heaps in video memory, instead requiring the application to allocate in such a way as to minimize fragmentation.

6.1.1.2.2 Allocating and Freeing Video Memory

This topic describes considerations for allocating and freeing video memory.

6.1.1.2.2.1 Allocating buffers

When using OpenGL ES/EGL, there is only a small set of APIs that actually lead to long-term video memory buffer allocation:

```
glBufferData(enum target, sizeiptr size, const void *data,
            enum usage)
glTexImage2D(enum target, int level, int internalFormat, sizei width,
            sizei height, int border, enum format, const void *pixels)
```

```
glCopyTexImage2D(enum target, int level, enum internalformat, int x,
int y, sizei width, sizei height, int border)
```

**Note:**

The `glCopyTexImage2D` function allocates only when it copies to a null.

```
eglCreateWindowSurface(EGLDisplay dpy, EGLConfig config,
NativeWindowType win, const EGLint *attrib_list)
```

```
eglCreatePbufferSurface(EGLDisplay dpy, EGLConfig config, const
EGLint *attrib_list)
```

```
eglCreatePixmapSurface(EGLDisplay dpy, EGLConfig config,
NativePixmapType pixmap, const EGLint *attrib_list)
```

6.1.1.2.2.2 Freeing buffers

A similar set of APIs free allocated video memory buffers, whether they are textures, VBOs, or surfaces:

```
glDeleteBuffers(sizei n, uint *buffers)
```

```
glDeleteTextures(sizei n const uint *textures)
```

```
eglDestroySurface(EGLDisplay dpy, EGLSurface surface)
```

**Note:**

The `glDeleteTextures` function works only if the texture object is greater than zero; the default texture can't be explicitly deleted (although it can be replaced with a texture containing one or two dimensions of zero, which accomplishes the same thing).

Conceptually, these calls can be thought of as `malloc()` and `free()` for VBOs and texture maps, respectively. The same techniques for avoiding fragmentation can also be applied.

6.1.1.2.2.3 Updating a Subregion of a Buffer

In many cases, avoiding fragmentation means placing multiple objects into the same shared buffer, or reusing a buffer by deleting or overwriting an older object with a newer one. OpenGL ES provides a method for updating an arbitrary section of allocated VBOs, textures, and surfaces:

```
glBufferSubData(enum target, intptr offset, sizeiptr size,
const void *data, enum usage)
```

```
glTexSubImage2D(enum target, int level, int xoffset, int yoffset,
sizei width, sizei height, enum format, enum type, const void *pixels)
```

```
glCopyTexSubImage2D(enum target, int level, int xoffset, int yoffset,
int x, int y, sizei width, sizei height)
```

```
glScissor(int left, int bottom, sizei width, sizei height);
```

```
glViewport(int x, int y, sizei w, sizei h)
```

The `glTexSubImage2D` and `glCopyTexSubImage2D` function update a subregion of the target texture image. In the first case, the source comes from an application buffer; in the second, from a rendering surface.

The `glScissor` and `glViewport` functions limit rendering to a subregion of a rendering surface. The first specifies the region of the buffer that the `glClear` function will affect; the second updates the transforms to limit rendered OpenGL ES primitives to the specified subregion.

6.1.1.2.2.4 Using a Buffer Subregion

Completing the functionality needed to reuse allocated buffers is the ability to use an arbitrary subregion of a texture, VBO, or surface:

```
glDrawArrays(enum mode, int first, sizei count)
glReadPixels(int x, int y, sizei width, sizei height, enum format, enum type, void
 *data)
glCopyTexImage2D(enum target, int level, int x, int y, sizei width, sizei height)
glCopyTexSubImage2D(enum target, int level, int xoffset, int yoffset, int x, int y,
 sizei width, sizei height)
```

For VBOs, the `glDrawArrays` function allows the application to choose a contiguous subset of a VBO.

For textures, there's no explicit call to limit texturing source to a particular subregion. But texture coordinates and wrapping modes can be specified in order to render an arbitrary subregion of texture object.

For surfaces, the `glReadPixels` function can be used to read from a subregion of a display, when copying data back to an application-allocated buffer.

The `glCopyTexImage2D` and `glCopyTexSubImage2D` functions also restrict themselves to copying from a subregion of the display surface when transferring data to a texture map. The only area that's problematic is controlling the direct display of window surface back buffer. OpenGL ES and EGL have no way to show only a subregion of the backbuffer surface, but the native windowing systems may have this functionality.

6.1.1.2.3 Best Practices for Video Memory Management

The following is a list of good practices when allocating video memory to avoid or minimize fragmentation:

6.1.1.2.3.1 1. Allocate large buffers early

Ideally allocate large buffers at the start of the program. On average, allocating large surfaces gets more difficult as more allocations occur. When more allocations occur, free space is broken into smaller pieces.

6.1.1.2.3.2 2. Combine many small allocations into a smaller number of larger allocations

Small allocations can disproportionately reduce available free space. Not only does the allocator have a fixed overhead per allocation, regardless of size, but small allocations tend to break up large areas of free space into smaller pieces.

The ability to load a subregion of a VBO or texture map, and the ability to render that subregion independently, makes it possible to combine VBOs and textures together. For textures, a large texture can be used to hold a grid of smaller images. For VBOs, multiple vertex arrays can be combined end to end into a larger one. Besides reducing fragmentation, combining related images into a single texture, and related vertex arrays into a single VBO often improves rendering time, since it reduces the number of `glBindBuffer` or `glBindTexture` calls required to render a set of related objects.

6.1.1.2.3.3 3. Reduce the variation in size of allocated buffers ideally to a single size

Allocating buffers of varying sizes, especially sizes that aren't small multiples of each other, is disruptive of memory space and causes fragmentation. The ability to load and render a subset of a VBO or texture means that data loaded doesn't have to match the size of the allocated buffer; as long as it's smaller, it will work.

This approach does waste space, in that some of the allocated buffer isn't used, but this waste is often offset by the saving in reduced fragmentation and fixed allocation overhead. This approach can often be combined with approach (2) (combining multiple objects into one allocated buffer) to reduce total wastage. Generally, it's safe to ignore wastage if it's a small percentage of the allocated buffer size (say < 5%).

This approach is particularly good for dynamically allocated data, since fixed size blocks can often be freed and reallocated with little or no fragmentation. If wastage is excessive, a set of buffer sizes can be chosen (often a consecutive set of power of two sizes), and the smallest free buffer that will contain the data is used.

6.1.1.2.3.4 4. Reuse, rather than free and reallocate, buffers whenever possible

The ability to reload a previously allocated buffer with new data for both VBOs and textures makes this technique practical. Reuse is particularly important for large buffers; it is often better to create a large buffer at program start, and reuse it during mode switches, etc., even if it requires allocating a larger buffer to handle all possible cases.

6.1.1.2.3.5 5. Minimize dynamic allocation

If possible, take memory allocation and freeing out of the inner loop of your application. The ability to reuse buffers makes this practical, even for algorithms that require dynamic allocation. Even with reuse, however, it's still better to organize the code to minimize allocations and frees, and to move the remaining ones out of the main code path as much as possible.

6.1.1.2.3.6 6. Try to group dynamic allocations

If dynamic allocation is mandatory, try to group similar allocations and frees together. Ideally, an allocation of a buffer is followed by freeing it before another allocation happens. This rarely can be done in practice, but combining a group of related allocations is often nearly as effective.

Again, allocations and frees should be replaced whenever possible. Grouping them is a last resort.

6.1.1.3 Graphics Driver CPU Usage

In some cases, the reported graphics driver CPU usage may be high, but in fact the yield is related to other CPUs. To reduce the reported CPU usage, set the environment variable as follows:

```
$ export __GL_YIELD=USLEEP
```

6.1.2 EGLStream

EGLStream is a mechanism that efficiently transfers a sequence of image frames from one API to another. APIs can be OpenGL, CUDA, or NvMedia. A producer and a consumer are attached to two ends of a stream object:

- A producer adds image frames into the stream.
- A consumer retrieves image frames from the stream.

6.1.2.1 EGLStream Producer

The EGLStream producer is the entity that posts EGL image frames into the EGLStream. The producer is responsible for inserting each image frame into the EGLStream at the correct time so that the consumer can display the image frame for the appropriate period of time.

Different types of producers:

- **CUDA producer:** Posts a CUDA array or CUDA pointer as EGL image frames to the EGLStream.
- **GL producer:** Posts graphic surfaces as EGL image frames to the EGLStream.

6.1.2.2 EGLStream Consumer

The EGLStream consumer is the entity that retrieves EGL image frames from the EGLStream. The consumer is responsible for noticing that an image frame is available and displaying it (or otherwise consuming it). The consumer is also responsible for indicating the latency when that is possible (the latency is the time that elapses between the time it is retrieved from the EGLStream until the time it is displayed).

Different types of consumers:

- **CUDA consumer:** Retrieves EGL image frames and fills the frame information as a CUDA array or CUDA pointer. The CUDA frames can then be processed in the CUDA domain.
- **GL consumer:** Retrieves EGL image frames that can then be bound as OpenGL textures for graphics rendering.

- **EGLOutput consumer (egldevice window system only)**: Retrieves EGL image frames and renders them directly to EGLOutput. This consumer is valid when EGLOutput is used on the egldevice window system.

6.1.2.3 EGLStream Operation Modes

There are two types of EGLStream operation modes: mailbox mode and FIFO mode.

6.1.2.3.1 Mailbox Mode

In mailbox mode, EGLStream conceptually operates as a mailbox. When the producer has a new image frame it empties the mailbox (discards the old contents) and inserts the new image frame into the mailbox. The consumer retrieves the image frame from the mailbox and examines it. When the consumer is finished examining the image frame, it is either placed back in the mailbox (if the mailbox is empty) or discarded (if the mailbox is not empty).

Timing is mainly controlled by the producer. The consumer operates with a fixed latency that it indicates to the producer through the EGLStream attribute EGL_CONSUMER_LATENCY_USEC_KHR. The consumer is expected to notice when a new image frame is available in the EGLStream, retrieve it, and display it in the time indicated by EGL_CONSUMER_LATENCY_USEC_KHR. The producer controls when the image frame is displayed by inserting it into the stream at time $T - EGL_CONSUMER_LATENCY_USEC_KHR$, where T is the time the image frame appears.

6.1.2.3.2 FIFO Mode

In FIFO mode no images are discarded. When a producer adds image frames to the stream, they are placed in a queue for subsequent retrieval by the consumer. EGLStream sets the queue size when it creates the queue. If the producer attempts to insert a frame and the FIFO queue is full, then the producer stalls until there is room in the FIFO queue. When the consumer retrieves an image frame from the EGLStream, it sees the image frame that immediately follows the image frame that it last retrieved (unless no such frame has been inserted yet, in which case it retrieves the same image frame that it retrieved last time).

Timing of an EGLStream in FIFO mode is the responsibility of the consumer. Each image frame in the FIFO has an associated timestamp set by the producer. The consumer uses this timestamp to determine when the image frame is intended to be displayed.

6.1.2.4 EGLStream Pipeline

EGL provides functions to create and destroy EGLStreams, to query and set attributes of EGLStreams, and to connect EGLStreams to producers and consumers.

Each EGLStream must be connected to only one producer and one consumer. Once an EGLStream is connected to a consumer, it is connected to that consumer until the EGLStream is destroyed. Likewise, once an EGLStream is connected to a producer, it is connected to that producer until the EGLStream is destroyed.

The EGLStream cannot be used until it has been connected to a consumer and then to a producer. It must be connected to a consumer before it is connected to a producer.

6.1.2.4.1 Building a Simple EGLStream Pipeline

Before you build a simple EGLStream pipeline, you must initialize the EGL interface on the platform and create the EGLDisplay.

1. Create the EGLDisplay object for the EGLStream to bind to it.
2. Call `eglInitialize()` to initialize EGL on the created display or on the default display.

If the GL consumer or EGLOutput consumer is used, you must initialize the rendering window before creating the EGLStream pipeline.

If the CUDA producer-consumer is used, window system initialization is not required.

3. Create an EGLStream.

Example for creating the EGLStream (from `eglstreamcube.c`):

```
client->stream = eglCreateStreamKHR(demoState.display, streamAttr);
if (client->stream == EGL_NO_STREAM_KHR) {
    NvGlDemoLog("Couldn't create EGL stream.\n");
    goto fail;
}
```

Depending on whether mailbox mode or FIFO mode is used, `streamAttr` is set accordingly. The attribute `EGL_STREAM_FIFO_LENGTH_KHR` is initialized for FIFO mode.

```
if (demoOptions.nFifo > 0) {
    streamAttr[numAttrs++] = EGL_STREAM_FIFO_LENGTH_KHR;
    streamAttr[numAttrs++] = demoOptions.nFifo;
}
```

4. Create a consumer and connect it to the EGLStream. It is specific to the consumer type.

Example for connecting a GL consumer to EGLStream (from `eglstreamcube.c`):

```
glBindTexture(GL_TEXTURE_EXTERNAL_OES, texture);
if (!eglStreamConsumerGLTextureExternalKHR(demoState.display, client->stream)) {
    NvGlDemoLog("Couldn't bind texture.\n");
    goto fail;
}
```

Once an EGLStream is connected to a consumer, it remains connected to the same consumer until the EGLStream is destroyed.

5. Create a producer and connect it to the EGLStream. It is specific to the producer type.

Example for connecting a GL producer to EGLStream (from `nvgldemo_main.c`):

```
eglCreateStreamProducerSurfaceKHR(demoState.display,
                                  demoState.config,
                                  demoState.stream,
                                  srfAttrs);
```

Once an EGLStream is connected to a producer, it must remain connected to the same producer until the EGLStream is destroyed.

6. The producer posts the image frame to the consumer, depending on the producer type.

In the GL producer case, `eglSwapBuffers` posts the buffer to the consumer.

7. The consumer acquires the image frame posted by the producer, uses it, and then releases the frame back to the stream. Methods for acquiring frames from a stream and releasing them back to a stream are dependent on the type of consumer.

Example for acquiring and releasing the frame in the GL consumer case (from `eglstreamcube.c`):

```
eglStreamConsumerAcquireKHR(demoState.display,
                             clientList[i].stream)
eglStreamConsumerReleaseKHR(demoState.display,
                           client->stream)
```

In the GL consumer case, If `eglStreamConsumerAcquireKHR()` is called twice on the same EGLStream without an intervening call to `eglStreamConsumerReleaseKHR()`, then `eglStreamConsumerReleaseKHR()` is implicitly called at the start of `eglStreamConsumerAcquireKHR()`.

6.1.2.4.2 Destroying the EGLStream Pipeline

Destroy the EGLStream pipeline in the following order:

1. Destroy the producer.
2. Destroy the consumer.
3. Destroy the EGLStream.
4. Destroy the window system resources.

6.1.2.4.3 EGLStream State

After an EGLStream is created, at any given time, the EGLStream is in one of the following defined states:

- > `EGL_STREAM_STATE_CREATED_KHR`: The EGLStream has been created but not yet connected to a producer or a consumer.
- > `EGL_STREAM_STATE_CONNECTING_KHR`: The EGLStream has been connected to a consumer but not yet connected to a producer.
- > `EGL_STREAM_STATE_EMPTY_KHR`: The EGLStream has been connected to a consumer and a producer, but the producer has not yet posted any image frames.
- > `EGL_STREAM_STATE_NEW_FRAME_AVAILABLE_KHR`: The producer has posted at least one image frame that the consumer has not yet acquired.
- > `EGL_STREAM_STATE_OLD_FRAME_AVAILABLE_KHR`: The producer has posted at least one image frame, and the consumer has acquired the most recently posted image frame.
- > `EGL_STREAM_STATE_DISCONNECTED_KHR`: The producer, the consumer, or both, are no longer connected to the EGLStream (e.g., they have been destroyed). Once the EGLStream is in this state it remains in this state until the EGLStream is destroyed.

In this state only `eglQueryStreamKHR()` and `eglDestroyStreamKHR()` are valid operations.

The EGLStream state is queried as follows:

```
EGLBoolean eglQueryStreamKHR(
    EGLDisplay dpy,
    EGLStreamKHR stream,
    GLenum attribute,
    EGLuint64KHR *value);
```

The following state transitions may occur:

1. EGL_STREAM_STATE_CREATED_KHR: A new EGLStream is created in this state.
2. EGL_STREAM_STATE_CREATED_KHR to EGL_STREAM_STATE_CONNECTING_KHR: Occurs when a consumer is connected to the EGLStream.
3. EGL_STREAM_STATE_CONNECTING_KHR to EGL_STREAM_STATE_EMPTY_KHR: Occurs when a producer is connected to the EGLStream.
4. EGL_STREAM_STATE_EMPTY_KHR to EGL_STREAM_STATE_NEW_FRAME_AVAILABLE_KHR: Occurs the first time the producer inserts an EGL image frame.
5. EGL_STREAM_STATE_NEW_FRAME_AVAILABLE_KHR to EGL_STREAM_STATE_OLD_FRAME_AVAILABLE_KHR: Occurs when the consumer begins acquiring a newly posted EGL image frame.
6. EGL_STREAM_STATE_OLD_FRAME_AVAILABLE_KHR to EGL_STREAM_STATE_NEW_FRAME_AVAILABLE_KHR: Occurs when the producer posts a new EGL image frame.
7. Any state to EGL_STREAM_STATE_DISCONNECTED_KHR: Occurs when the producer or consumer is destroyed.

6.1.2.5 Building a Cross-Process EGLStream Pipeline

A cross-process EGLStream is one where the producer and consumer are in different processes.

NVIDIA provides two options for establishing a cross-process stream. One uses the `EGL_KHR_stream_cross_process_fd` extension, and the other uses the `EGL_NV_stream_remote` extension. In both cases, producer and consumer processes are responsible for establishing a socket connection with each other before beginning to create the stream.

With the `EGL_KHR_stream_cross_process_fd` extension, one of the applications creates the EGLStream and then gets a file descriptor from it. In the sample code, the consumer creates the stream, but it could be either end. The creator then obtains a file descriptor from the stream and passes it over to the socket to the other process. That process receives the file descriptor and uses it to create the other end of the EGLStream. After both endpoints have created their EGLStream object, consumer and producer connection proceeds as in the single process case. The processes maintain ownership of the socket, and can continue to use it for any other communication they need.

With the EGL_NV_stream_remote extension, the processes may use the sockets they create for any initial handshaking and communication, but then pass ownership of them to the stream. Producer and consumer both create EGLStream endpoints from their socket. For cross-process streams within the same partition, NVIDIA requires that the type used for the socket in the EGLStream creation call be UNIX. INET sockets are not supported. (This is in contrast with cross-partition streams described below.)

You can execute the following cross-process examples in two shells or run one in the background in one shell.

6.1.2.5.1 Cross-process EGLStream example

Consult the samples/opengles2/eglstreamcube and samples/opengles2/bubble examples. eglstreamcube is the consumer, and bubble app is the producer.

6.1.2.5.1.1 Consumer-side steps:

1. Create the stream and get the file descriptor of the stream (refer to eglstreamcube.c).

```
client->stream = eglCreateStreamKHR(demoState.display,      streamAttr);
client->fd = eglGetStreamFileDescriptorKHR(demoState.display, client->stream);
```

2. Share the file descriptor with the producer through the socket.
 3. Bind the consumer end of the EGLStream with the GL texture (refer to eglstreamcube.c).
- ```
glBindTexture(GL_TEXTURE_EXTERNAL_OES, texture);
eglStreamConsumerGLTextureExternalKHR(demoState.display, client->stream)
```
4. Latch the recent image frame to the texture with eglStreamConsumerAcquireKHR:
- ```
eglStreamConsumerAcquireKHR(demoState.display, stream)
```
5. Render the texture as one face of the cube.

6.1.2.5.1.2 Producer-side steps:

1. Receive the stream file descriptor through the socket (given as an eglstreamsocket argument) and create the stream (refer to nvldemo_main.c):

```
eglCreateStreamFromFileDescriptorKHR(demoState.display, fd)
```

2. Create the EGLStream surface (refer to nvldemo_main.c).

```
eglCreateStreamProducerSurfaceKHR(
    demoState.display,
    demoState.config,
    demoState.stream,
    srfAttrs)
```

3. Create EGLContext and eglMakeCurrent on the EGLStream surface created in step 2.
4. Start to render the frames and eglSwapBuffers.

6.1.2.5.1.3 Run the cross-process example

```
./eglstreamcube -dispno 1 -layer 1 -windowoffset 1000 0
-socket /tmp/test &
./bubble -eglstreamsocket /tmp/test &
```

The bubble app content appears on one face of the cube.

6.1.2.6 Building a Cross-Partition EGLStream Pipeline

A cross-partition EGLStream is one where the producer and consumer are in two different partitions.

Creating a cross-partition stream is similar to the second method (using `EGL_NV_stream_remote`) for creating a cross-process extension described above. The two processes must establish socket communication with each other, and then pass ownership of that socket to the stream, with each end creating an EGLStream object from their socket. For cross-partition streams, NVIDIA requires that the type used for the socket be INET. (This is in contrast with cross-process streams described above.)

In our example code, the consumer acts as a server, opening a socket and listening for a connection on a known address. The producer then connects to that address, and both sides obtain the socket to use for the stream. They then proceed to create their EGLStreams. After both endpoints have created their EGLStream object, consumer and producer connection proceeds as in the single process case.

You can execute the following cross-partition examples in two shells, one for each partition.

To run cross-partition samples, `hv0` of the consumer and producer partition must be configured to use two different IP addresses. For example, on VM1 (the producer partition), set it to 12.0.0.2 using:

```
ifconfig hv0 12.0.0.2
```

Similarly, on the consumer partition, set it to 12.0.0.1.

6.1.2.6.1 Cross-partition EGLStream example

Consult the `samples/opengles2/eglstreamcrosspart` example.

6.1.2.6.1.1 Consumer-side steps:

1. Create the socket on the port number given as an argument (default: 8888) and set the created socket ID to `g_ServerID` (refer to `nvgldemo_main.c`).
2. Create the cross-partition EGLStream for the consumer end by setting the attributes (refer to `nvgldemo_main.c`).

```
EGL_STREAM_TYPE_NV:EGL_STREAM_CROSS_PARTITION_NV,
EGL_STREAM_ENDPOINT_NV:EGL_STREAM_PRODUCER_NV,
EGL_SOCKET_HANDLE_NV: g_SEGLerverID
demoState.stream = eglCreateStreamKHR(demoState.display, attr);
```

3. Bind the consumer end of the EGLStream with the GL texture (refer to `eglstreamcrosspart/consumer.cpp`).

```
glBindTexture(GL_TEXTURE_EXTERNAL_OES, args.videoTexID);
eglStreamConsumerGLTextureExternalKHR(args.display,
args.eglStream)
```

4. Latch the recent image frame to the texture with `eglStreamConsumerAcquireKHR`, render the frame, and call `eglSwapBuffers` (refer to `eglstreamcrosspart/consumer.cpp`):

```
eglStreamConsumerAcquireKHR(args.display, args.eglStream)
```

6.1.2.6.1.2 Producer-side steps:

1. Connect to the socket with the given IP address and the port number arguments. Set the created socketID to `g_ClientID` (refer to `nvgldemo_main.c`)
2. Create the cross-partition EGLStream for the producer end by setting the attributes (refer to `nvgldemo_main.c`):

```
EGL_STREAM_TYPE_NV:EGL_STREAM_CROSS_PARTITION_NV,
EGL_STREAM_ENDPOINT_NV:EGL_STREAM_PRODUCER_NV,
EGL_SOCKET_HANDLE_NV: g_ClientID
```

3. Create the EGLStream surface (refer to `nvgldemo_main.c`):

```
eglCreateStreamProducerSurfaceKHR(
    demoState.display,
    demoState.config,
    demoState.stream,
    srfAttrs)
```

4. Create EGLContext and `eglMakeCurrent` the EGLStream surface created in step 2 (refer to `nvgldemo_main.c`).
5. Start to render the frames and `eglSwapBuffers` (refer to `eglstreamcrosspart/producer.cpp`).

6.1.2.6.1.3 Run the cross-partition sample

On Linux, run EGLStream producer:

```
/samples/opengles2/eglcrosspart/egldevice/eglcrosspart
 -proctype producer -ip 12.0.0.1 -port 1111 &
```

6.1.3 Binary Shader Program Management

Pre-built binary shader programs eliminate compilation time for individual shaders. If all programs are pre-built, the driver may avoid consuming additional time and resources by not loading the compiler libraries at all. Program binaries can be compiled and linked by an application calling the OpenGL ES API directly at runtime or prebuilt with the `glslc` offline shader compiler.

The NVIDIA® SDK no longer supports the `GL_NV_platform_binary` extension for use with the OpenGL ES 2.0 `glShaderBinary` function. Program binaries can be saved and loaded either by an application calling the OpenGL ES API directly, or using the automatic shader cache.

6.1.3.1 Application Management of Binary Programs

Applications can control binary program management directly. With the `glGetProgramBinary` function added in OpenGL ES 3.0, applications can read back

the binaries for any compiled and linked programs, and store them for later use. The `glProgramBinary` function loads these saved binaries. Unlike the `glShaderBinary` function in OpenGL ES 2.0, this operates on a linked program instead of individual vertex and fragment shaders, so there is no additional link step required after loading the binary.

6.1.3.1.1 Automatic Shader Cache

When the proprietary NVIDIA shader cache support in the driver is enabled, the OpenGL ES 3.0 driver maintains a shader cache file for each program. When an application specifies a shader source, the driver first searches the cache to see if it has already compiled this source with the current version of the compiler. If not, it compiles the program and then saves a copy in the cache. If the shader has been previously compiled, the driver loads the pre-built binary from the cache. The cache persists between application runs, so that with a thorough initial test run, compilation need only occur the first time a new driver is installed.

The shader cache is enabled on read/write file systems by default. Assign the `__GL_SHADER_DISK_CACHE` environment variable a value of 0 to disable the cache.



Note:

The shader cache is disabled by default on QNX as the root file system is read only. The cache can be enabled by setting the environment variables `__GL_SHADER_DISK_CACHE` to 1 and `__GL_SHADER_DISK_CACHE_PATH` to a location with write access.

By default, the cache is placed in an `.nv` directory in the user's home directory, which is created if it does not already exist. If desired, override the default location by specifying a valid alternate path with the `__GL_SHADER_DISK_CACHE_PATH` environment variable. Unlike the `.nv` directory, the driver does not create this non-default directory if it does not already exist. If this variable specifies a directory that does not already exist, the cache is disabled.

The cache appears in the specified location as a subdirectory named `GLCache` containing a set of directories and files with hashed names. These files encode the driver version, GPU, and application. Whenever you install a new version of the driver, the hashed names change. Users may therefore wish to clean out the old directory before running any applications.

6.1.3.1.2 To use a read-only cache

1. Run the application on a test/development system that has a file system with write access.
2. Copy the cache directory to the desired location in the target image.
3. Set the environment variables described above to enable the cache and indicate its location.

6.1.3.2 Comparison and Combination

The main advantage offered by the shader cache is that it is handled automatically, without any application intervention. In addition, the driver sometimes needs to generate shaders internally for certain clear and copy operations. If the cache is enabled, these too will only be generated once, rather than every time the application is run.

Although the cache eliminates the need to recompile shaders, there is search and maintenance overhead. Applications can avoid this overhead by saving and loading programs directly. Furthermore, the cache files become invalid every time a new driver is installed. Saved binaries, on the other hand, only need to be replaced when the compiler portion of the driver is updated.

The shader cache and the functions to read and load binaries are not mutually exclusive. An application can make use of both. For instance, the most critical shader programs which must be available as soon as possible after startup could be manually saved, while less frequently used shaders rely on the cache.

6.1.4 GLSLC Shader Program Compiler

This section describes `glslc`, a compiler for OpenGL ES 3.0-style program binaries. This compiler runs on the Linux host system to produce program binaries that can be transferred to the target NVIDIA Tegra device.



Note:

Program binaries produced with `glslc` from a particular NVIDIA DRIVE[®] SDK can only be used with the OpenGL ES 3.0 driver from the same NVIDIA DRIVE[®] SDK. Shader sources must be recompiled to generate new program binaries whenever a new SDK is installed.

To compile a shader program

- > Generate a program binary `<output_file>` from vertex and fragment shader source files `<vertex_shader_file>` and `<fragment_shader_file>` with the following command (on the host system):

```
glslc -gles -chip 10 -binary <output_file> -vs <vertex_shader_file> -fs
<fragment_shader_file>
```

Shader sources must not contain `#include` directives. If `glslc` is successful, it produces a non-zero sized binary named `<output_file>`.



Note:

For complete information about additional shader types and supported options for `glslc`, run the following command:

```
glslc --help
```

6.1.4.1 Compiled Shader Program Characteristics

The resulting program binary only functions correctly with the version of the driver with which the compiler was released.

The format of a compiled program binary is:

- > 4 bytes containing the size of the program binary
- > 4 bytes containing the format of the program binary
- > A variable number of bytes containing the program data

These should be passed as the 4th, 2nd, and 3rd arguments, respectively, of `glProgramBinary`. See the `gearslib.c` file for more details on loading shader programs.

After a successful compilation, `glslc` displays the following message:

```
Program binary successfully written to <binary file name>
```

The size and format of a program binary produced by `glslc` are expected to match the size and format of a program binary produced by the GL driver via a `glGetProgramBinary` call.

6.1.4.2 Libraries Loaded On-Demand

On supported systems, these libraries are only loaded when needed to compile shader programs at runtime:

```
libnvidia-glcore-cg.so.<version>
libnvidia-glcore-ocg.so.<version>
```

If all of the program binaries used by the application are precompiled, the OpenGL ES driver does not invoke the compiler, and the libraries are not loaded. Also, if the system supports the shader disk cache and the required shaders are found there (usually after the first time the application is run), the libraries are not loaded.



Note:

Systems supporting the on-demand loading of these libraries are the systems used with *NVIDIA DRIVE Linux SDK* and *NVIDIA DRIVE QNX SDK*. See the *Release Notes* for those releases for specific hardware and software information.

The `-driverstate` option allows for a small subset of GPU machine microcode to be inserted into the GL program binary via `glProgramBinary`. In very specific and limited cases, this option can prevent `libnvidia-glcore-ocg.so` from loading at runtime. GL may optimize machine code depending on the driver state, requiring a new machine code binary to be generated depending on certain states of the driver for each draw call.

A different `-driverstate` parameter must be specified for each draw call or `glClear` call. Multiple `-driverstate` commands can be input in a single GLSLC command line. The resulting microcode is appended to the same program binary file.

As an example, consider an application utilizing vertex attribute arrays that makes two draw calls, where the state of those arrays changes between each draw call. If attribute

array 0 is enabled and attribute array 1 is disabled (constant attribute) during the first draw call, and both arrays are enabled during the second draw call, then the GLSLC command to obtain the program binary with optimized GPU machine code for both these states is similar to the following:

```
glslc -chip 10 -vs vs.vert -fs fs.frag
      -binary prog.bin -driverstate vertexattribenable 0 -driverstate
      vertexattribenable 0 vertexattribenable 1
```

The two `-driverstate` flags correspond to the state of the driver during each draw call, and there may be multiple `vertexattribenable` flags for each `-driverstate` usage.

A default set of machine code configurations are included if you use any `-driverstate` flags. Specify an empty `-driverstate` option followed by no state options if you require a set of default machine code configurations.

For example, the following produces a binary with a default set of machine code:

```
glslc -chip 10 -vs vert.vs -fs frag.fs
      -driverstate
```

You do not need an empty `-driverstate` option to include the defaults. The defaults are included if you use a `-driverstate` flag of any kind.

For example, the first `-driverstate` option below is redundant:

```
glslc -chip 10 -vs vert.vs -fs frag.fs
      -driverstate -driverstate vertexattribenable 0
```

The following is all that is needed:

```
glslc -chip 10 -vs vert.vs -fs frag.fs
      -driverstate vertexattribenable 0
```



Note:

The available options are currently limited to vertex and fragment shaders with a limited subset of options for producing the machine code. If the required microcode is not a part of the program binary during runtime, then the `libnvidia-glcore-ocg.so` library is loaded to perform runtime compilation.

6.1.5 Tegra GPU Scheduling Improvements



Warning:

There is a conflict in environment variables when using the same terminal to build and flash the tools. Unset the `TEGRA_TOP` environmental variable before trying to flash, or use different terminals to build and flash.

NVIDIA[®] Tegra[®] GPU scheduling architecture addresses the quality of service (QoS) for several classes of work. The following table summarizes these classes of work.

Priority	Description	Workload
HIGH	Critical applications that must meet their rendering deadlines	Small workloads (executable within a display refresh cycle), typically 60 frames/second (fps)
MEDIUM	Well-behaved applications that are not known to cause channel reset	Small-to-medium workloads (may be spread across several refresh cycles)
LOW	Long-running or potentially rogue applications (e.g., WebGL contexts)	Small-to-large workloads

Scheduling Parameters

Applications can achieve the desired QoS by tweaking the following scheduling parameters:

- **timeslice**: Specifies the maximum time a channel can use an engine uninterrupted.
- **runlist_interleave_frequency**: Specifies the number of times a channel can appear on a runlist.
- **preemption_type**: Defines the preemption boundary and how a context is saved.

Previous Limitations

In the previous implementation, applications could set the timeslice (via a sysfs interface) and the preemption type, but the runlist interleave frequency was fixed at 1. This resulted in high-priority applications receiving only one scheduling point per iteration of all channels on the runlist. (For more information, see [Runlist Interleave Frequency](#) in this chapter). This implementation was insufficient to let high-priority applications reach their target frame rate.

6.1.5.1 Setting the Timeslice

Use the following guidelines when setting the timeslice, depending on the priority of the application.

High-Priority Applications

For high priority applications, set the timeslice large enough so that all work can be completed within one timeslice.

The recommended upper bound for timeslice in single, high-priority applications is:

$$16.6 \text{ ms} - lpt - cst - crt = 11.6 \text{ ms}$$

Where:

- *lpt* is the low-priority timeslice, set to 1.5 ms
- *cst* is the context-switch timeout, equal to 2.0 ms
- *crt* is the channel reset time, equal to 1.5 ms

For multiple, high-priority applications, use the timeslice for each high-priority application to determine a reasonable bound. The recommended combined workload of all high-priority applications must not exceed 50% of a display refresh cycle.

High-priority applications must avoid flushing work prematurely, whether by calling `glFlush` or `glFinish` or by other means. This ensures all rendering for a frame completes without any context switches.

Medium-Priority Applications

For medium-priority applications, set the timeslice both:

- > Large enough that an application can make progress, but
- > Not so large that it affects the scheduling latency of high-priority applications.

The recommended upper bound for timeslice in medium-priority applications is 2 ms.

Low-Priority Applications

For low-priority applications, set the timeslice both:

- > Large enough that an application can make progress, but
- > Not so large that it affects the scheduling latency of high- or medium-priority applications.

The recommended upper bound for timeslice in low-priority applications is 1.5 milliseconds (ms).

Reserve Time for Lower-Priority Applications

To ensure lower-priority applications make reasonable progress, you must ensure that high- and medium-priority applications do not use 100% of the GPU by:

- > Lowering your application frame rate targets and/or
- > Reducing complexity of rendered frames.

The proportion of time to reserve for low-priority applications depends on the number and nature of the applications.

6.1.5.2 Setting the Preemption Type

High-Priority Applications

For high-priority applications, set the timeslice large enough that all work can complete. The recommended Compute-Instruction-Level-Preemption (CILP) setting for graphics and for compute is a preemption type of Wait-For-Idle (WFI). This ensures CILP will not be hit because NVIDIA CUDA kernels will have completed.

Medium-Priority Applications

The recommended setting for medium-priority applications is a preemption type enabled for graphics (GFXP) and compute (CILP). For applications that can complete in their timeslice, context-switch overhead is minimal because the GPU is in an idled state.

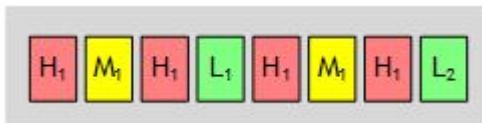
Low-Priority Applications

For low-priority applications, always enable graphics and compute preemption because workloads are unpredictable.

6.1.5.3 Runlist Interleave Frequency

The runlist is an ordered list of channels that the GPU HOST reads to find work for the downstream engines to complete. To enable the GPU HOST to schedule a given channel more often, include the channel multiple times on a runlist. For each priority level, the runlist interleave frequency must be set to match the priority.

For example, if a system has one high-priority application, one medium-priority application, and two low-priority applications, the GPU scheduler constructs the runlist as follows:



The scheduling latency for when a high-priority application will be able to run is governed by:

$$\text{worst-case latency(high)} = (h-1) \times \text{timeslice(high)} + \text{execution time(low)} + \text{channel reset}$$

Where:



6.1.5.4 Setting Parameters

Use the following guidelines when setting scheduling parameters.

- Identify the relevant use cases before setting scheduling parameters.
- Associate a priority with each application.
- Determine appropriate timeslices.

Then using the environment variables listed below, specify the runlist interleave frequency, timeslice, and preemption type.

Environment Variable	Role	Values
NVRM_GPU_CHANNEL_ INTERLEAVE	Sets runlist frequency for a context	1: LOW 2: MEDIUM 3: HIGH
NVRM_GPU_CHANNEL_ TIMESLICE	Sets timeslice for a context	Non-zero value in microseconds (minimum 1000 for 1 ms)
NVRM_GPU_NVGPU_FORCE_ GFX_ PREEMPTION	Enables GFXP	0: off 1: on

**Note:**

The environment variables apply to *all* contexts belonging to a process.

6.1.5.5 Setting Parameters on Behalf of Other Applications

A privileged application can set scheduling parameters (timeslice and interleave) on behalf of other applications based on their PID.

The `libnvrm_gpusched` library provides a way to:

- > Get a list of all TSGs
- > Get a list of recent TSGs (i.e., a list of TSGs opened since the last query)
- > Get a list of TSGs opened by a given process
- > Get notifications when a TSG is allocated
- > Get current scheduling parameters for a TSG
- > Set runlist interleave for a TSG
- > Set timeslice for a TSG
- > Lock control (i.e., prevent other applications from changing their own scheduling parameters such as timeslice and interleave)

Library API is defined in `nvrm_gpusched.h` and sample code implements command line to control GPU scheduling parameters.

6.1.5.6 Building GPU Scheduling Sample Applications

Enter:

```
cd <top>/drive-linux/samples/nvrm/gpusched
make clean
```

```
make
```

6.1.5.7 Running the GPU Scheduling Sample Application

Here is a step-by-step example that shows how to tweak the GPU scheduling parameters on behalf of other applications based on their PID.

To schedule a high-priority app to render at the desired frame rate

1. Boot native Linux or a single Linux VM with a 4K HDMI display supporting 60 Hz refresh rate attached to a NVIDIA DRIVE OS 6.0 Linux platform, and start X.

```
sudo ./nvrm_gpusched set interleave 3
sudo ./nvrm_gpusched set timeslice 11600
sudo -b X -ac -noreset -nolisten tcp
```

2. Start 16 instances of bubble by entering this command:

```
export DISPLAY=:0
export NVRM_GPU_NVGPU_FORCE_GFX_PREEMPTION=1
export NVRM_GPU_CHANNEL_INTERLEAVE=1
export NVRM_GPU_CHANNEL_TIMESLICE=1000
export NV_SWAPINTERVAL=1
export WIDTH=960
export HEIGHT=540
let Y=2160-HEIGHT
while [ $Y -ge 0 ]; do
let X=3840-WIDTH
while [ $X -ge 0 ]; do
<top>/drive-linux/samples/opengles2/bubble/x11/bubble -windowsize $WIDTH $HEIGHT -
windowoffset $X $Y -msaa 8 -fps &
sleep 1
let X=X-WIDTH
done
let Y=Y-HEIGHT
done
```

3. Wait about 10 seconds and record the frame rate of the bubble instances. All instances should have a similar frame rate, and lower than 60 fps.

4. Check scheduling parameters using:

```
sudo ./nvrm_gpusched get params
```

5. Change the timeslice and runlist interleave of the last instance by entering:

```
sudo ./nvrm_gpusched set timeslice -p <pid> 3000
sudo ./nvrm_gpusched set interleave -p <pid> 2
```

6. Wait about 10 seconds and record the frame rate of the last instance of bubble. It should be fixed around 60 fps, and the other instances should be running lower.

Note that NV_SWAPINTERVAL=1 limits the frame rate to the monitor's refresh rate.

7. Restore timeslice to its previous value for the last bubble instance:

```
sudo ./nvrm_gpusched set timeslice -p <pid> 1000
sudo ./nvrm_gpusched set interleave -p <pid> 1
```

8. Wait about 10 seconds and check that all bubble instances have about the same fps.

6.1.6 Disabling GPU Debugger and Profiler for Security



Important: Points to consider before using GPU Debugger and Profiler.

- > NVIDIA recommends disabling the GPU Debugger and Profiler support before deploying NVIDIA DRIVE OS or when the support is not required during development.



Note: The GPU Debugger and Profiler support is enabled by default in NVIDIA DRIVE OS to better development experience.

- > GPU Debugger and Profiler can examine and alter the state of all the applications running on the GPU, and attackers can exploit this capability.
- > Potential security risks could occur when this support is enabled during the DRIVE OS deployment.

To disable GPU Debugger and Profiler:

1. Identify the Guest OS DTB file getting flashed on the target.
2. Back up the original DTB file, and convert it to the DTS format with this command:
`dtc -I dtb -O dts <DTB> -o edit.dts`
3. Edit edit.dts and set the support-gpu-tools device tree property in the GPU device node to 0.
 - > For more information, see [<top>/kernel/kernel-5.10/Documentation/devicetree/bindings/gpu/nvidia_gv11b.txt](#).
 - > On NVIDIA DRIVE Orin™, the GPU device node name is ga10b.
4. Save the edits and compile DTS back to DTB format with command:
`dtc -I dts -O dtb edit.dts -o <DTB>`
5. Bind and flash the target as usual.

The GPU Debugger and Profiler support is disabled now.

To enable GPU Debugger and Profiler:

1. Back up the original DTB file, and convert it to the DTS format with this command:
`dtc -I dtb -O dts <DTB> -o edit.dts`
2. Edit edit.dts and set the support-gpu-tools device tree property in the GPU device node to 1.
 - > For more information, see [<top>/kernel/kernel-5.10/Documentation/devicetree/bindings/gpu/nvidia_gv11b.txt](#).
 - > On NVIDIA DRIVE Orin, the GPU device node name is ga10b.
3. Save the edits and compile DTS back to DTB format with command:
`dtc -I dts -O dtb edit.dts -o <DTB>`
4. Bind and flash the target as usual.

The GPU Debugger and Profiler support is enabled now.

6.1.7 Building and Running Samples

Samples must be built on the host. In order to run the samples, you must be logged onto the target system.

6.1.7.1 Building the OpenGL ES 2.0 Samples

The samples are provided with source code and a Makefile.

6.1.7.1.1 To Build the Samples

1. Make a copy of the default binaries.
2. On the host system, execute these commands:

```
cd <top>/drive-linux/samples/opengles2/<sample>
make clean
make
```

Where <sample> is the name of the sample that you want to build, for example, gears. By default, the preceding commands build X11 supported samples. For other windowing system supported samples, see the [Using NV_WINSYS when Building Graphics Samples](#) topic.

6.1.7.2 Using NV_WINSYS when Building Graphics Samples

The platform includes a default window system, but additional window systems are also supported. Each of the graphics samples may be built for all the supported window systems for a given platform.

6.1.7.2.1 To switch to a different window system

Export NV_WINSYS with one of these values:

```
egldevice
wayland
x11
direct-to-display (For Vulkan samples only)
```

Binaries are saved in separate subdirectories depending on the window system with the same name, so you can maintain the output for multiple window systems.

For example, building gears for all the window systems results in the following directories being populated with gears executables:

```
<top>/samples/opengles2/gears/<window_system>/gears
```

6.1.7.3 Copying the OpenGL ES 2.0 Samples to the Target FS

Run the copytarget script, passing in the path to the targetfs to ensure that the new binary is copied to the target:

```
<top>/drive-linux/utils/scripts/copytarget-samples <targetfs_dir>
```

6.1.7.4 Running the OpenGL ES 2.0 Samples

By default, if no command line arguments are supplied, the sample applications run at a native resolution for the connected display.

The sample apps can be run directly on EGLDevice, or on top of a window system. To run the samples on top of a window system, the window system must first be started. Supported window systems include:

- > [Wayland](#)
- > [X11](#)

For operating instructions, click on the link for your desired window system.

Using the egldevice version of gears as an example, run the sample with the following commands:

```
cd /home/nvidia/drive-linux/samples/opengles2/gears/egldevice
./gears
```

6.1.7.4.1 Resolution Selection

Set the application's output resolution with the following flag:

```
-windowsize <xres> <yres>
```

This is the window size of the application.

Set the screen resolution with:

```
-screensize <xres> <yres>
```

Use this option to set the screen size when there is no window system running, as is the case for egldevice. When a window system like X11 or Weston is running, the screen size is already set, and this option is ignored.

To specify a desktop offset, use the flag:

```
-windowoffset <xoffset> <yoffset>
```

6.1.7.4.2 Layer Selection

When running egldevice versions of the graphics samples, applications can select the overlay where to direct the sample output. Only one application can run per overlay, and the number of overlays per display head varies depending on the hardware configuration.

```
-layer <depth>
```

For example, on NVIDIA DRIVE AGX™ Platform, valid values are -layer [0|1] for -dispno [0|1] and -layer 0 for -dispno [2|3].

Here, dispno is the display number that is assigned serially from 0 through 3 for the four displays on a NVIDIA DRIVE AGX Orin™ Reference Board.

6.1.7.4.3 Antialiasing Specification

Specify multi-sample antialiasing (MSAA) with:

```
-msaa <numsamples>
```

The `<numsamples>` specified are platform dependent.

Valid values are 2, 4, or 8.

6.1.7.4.4 Program Binary Selection

Applications use this option to load the pre-compiled shader program binary from its current execution path. The program binaries are generated for all supported architectures when the `make` command is run. These binaries are generated into separate directories, each with the supported architecture name. The correct directory name needs to be specified when running the sample app.

If the program binaries are not present, the application creates the program binary from shader source and saves the created program binary in the current execution path. The application can then use these saved program binaries on the next run. The flag to control use of program binary is:

```
-useprogbin <boolean>
```

Where `<boolean>` is either 0 or 1.

6.1.7.4.5 Display Selection

When running `egldevice` versions of the graphics samples, applications can select the output device with the command:

```
-dispno <number>
```

Possible values for `<number>` are 0, 1, 2, etc. `<number>` corresponds to a display node. For example:

- `-dispno 0`: sets the display on `/dev/tegra_dc_0`
- `-dispno 1`: sets the display on `/dev/tegra_dc_1`
- `-dispno 2`: sets the display on `/dev/tegra_dc_2`

Each `/dev/tegra_dc_<number>` can be a different display type.

6.1.7.5 Running the Vulkan Samples

Direct-to-display WSI, Wayland, and X11/XCB are the supported backends for the Vulkan samples.

When running the Vulkan samples, you might need to temporarily disable other window systems. For example, with X11 installations, run these commands:

```
sudo service gdm3 stop
<run sample>
sudo service gdm3 start
```

There are two Vulkan sample applications, `vkfish` and `vkcube`, which must be compiled in the host and copied to the target. For compilation steps, see [Building and Running Samples](#).

The sample path for `vkfish`:

```
$PDK_TOP/drive-linux/samples/vulkan/vkfish
```

The sample path for `vkcube`:

```
$PDK_TOP/drive-linux/samples/vulkan/vkcube
```

Running `vkfish`

The `vkfish` application is a modified version of the NVIDIA threaded rendering Vulkan sample. It has been modified to use the direct-to-display WSI and Wayland platforms, and the interface has been stripped for simplification.

To run `vkfish` with direct-to-display:

1. Disable other window systems.
2. Run the following commands:

```
cd $SAMPLE_APP/vkfish/direct-to-display
./vkfish
```

`$SAMPLE_APP` represents the application directory path of the target.

3. Restart window systems, if necessary.

To run `vkfish` with Wayland:

1. Disable other window systems.
2. Install the driver module as follows:

```
sudo insmod /lib/modules/$(uname -r)/extra/opensrc-disp/nvidia-drm.ko modeset=1
```

3. Start Weston Wayland:

```
mkdir /tmp/xdg
chmod 700 /tmp/xdg
export XDG_RUNTIME_DIR=/tmp/xdg
weston-launch&
```

4. Run the following commands:

```
cd $SAMPLE_APP/vkfish/wayland
./vkfish
```

`vkfish` does not support command-line arguments and runs forever by default.

To run `vkfish` with X11:

1. Disable other window systems.

2. Start X11 as follows:

```
export DISPLAY=:0.0
sudo -b X -ac -noreset -nolisten tcp
```

3. Run the following commands:

```
cd $SAMPLE_APP/vkfish/x11
./vkfish
```

The sample might be run with `--c <framecount>` to render a certain number of frames. If started without arguments, the app will run forever.

Running vkcube

The vkcube application is a modified version of the LunarG Vulkan cube demonstration available at:

<https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers/tree/master/demos>

It has been modified to use the direct-to-display WSI and Wayland platforms.

To run vkcube with direct-to-display:

1. Disable other window systems.
2. Run the following commands:

```
cd $SAMPLE_APP/vkcube/direct-to-display
./vkcube
```

`$SAMPLE_APP` represents the application directory path of the target.

3. Restart window systems, if necessary.

The sample might be run with `--c <framecount>` to render a certain number of frames. If started without arguments, the app will run forever.

To run vkcube with Wayland:

1. Disable other window systems.
2. Install the driver module as follows:

```
sudo insmod /lib/modules/$(uname -r)/extra/opensrc-disp/nvidia-drm.ko modeset=1
```

3. Start Weston Wayland:

```
mkdir /tmp/xdg
chmod 700 /tmp/xdg
export XDG_RUNTIME_DIR=/tmp/xdg
weston-launch&
```

4. Run the following commands:

```
cd $SAMPLE_APP/vkcube/wayland
./vkcube
```

The sample might be run with `--c <framecount>` to render a certain number of frames. If started without arguments, the app will run forever.

To run vkcube with X11:

1. Disable other window systems.
2. Start X11 as follows:

```
export DISPLAY=:0.0
sudo -b X -ac -noreset -nolisten tcp
```

3. Run the following commands:

```
cd $SAMPLE_APP/vkcube/x11
```

```
./vkcube
```

The sample might be run with `--c <framecount>` to render a certain number of frames. If started without arguments, the app will run forever.

6.1.7.6 Vulkan SC Samples

Vulkan® SC samples can be built on both the host and the target by using the `cmake` command version 3.18 or later. The following topics describe how to build and run the Vulkan SC samples.



Note: The Vulkan SC samples and instructions are only supported on the Standard build.

6.1.7.6.1 Building the Vulkan SC Samples

To build the Vulkan® SC samples, you can use cross compilation, building on target, or host x86 binary build.

Prerequisites:

- > The `cmake` command version 3.18 or later
- > The `pkg-config` tool
- > The `build-essential` Linux package

To install, run the following command:

```
sudo apt-get install build-essential
```

Cross-Compiling

Cross compilation is supported only on a Linux host, such as desktop Linux (x86 Linux). To build the samples using cross compilation, compile in the PDK/SDK directory with the toolchain installed because all the required dependencies are in the PDK/SDK directory and the `cmake` command in the samples uses the relative path to the root of SDK/PDK.

On a Linux host system, use the following commands to build the samples:

```
cd <NV_WORKSPACE>/drive-linux/samples/vulkan-sc-samples
mkdir build
cd build
cmake -DCMAKE_TOOLCHAIN_FILE=../cmake/embedded-linux.cmake ../
make
```

After successful build, copy the `build/bin` and `vulkan-sc-samples/external/ktx` folders to the target device.

Building on the Target

To build the Vulkan SC samples on the target, copy the `vulkan-sc-samples` directory to the target and then build the samples on the target by using the following commands:

```
cd vulkan-sc-samples
mkdir build
```

```
cd build
cmake ../
make
```

Binaries Description

After the build, the following binaries are generated under the build/bin directory:

- > vksc_01tri
- > vk_01tri
- > vksc_computeparticles
- > vk_computeparticles
- > vulkanscinfo

Among those binaries, vksc_01tri, vksc_computeparticles, and vulkanscinfo are VulkanSC samples, while vk_01tri and vk_computeparticles are Vulkan samples. Because Vulkan SC API supports only offline pipeline cache, you must generate the pipeline cache for each sample. The Pipeline Cache Compiler (PCC) tool is used to generate the pipeline cache from the JSON files that describe the pipeline configurations. To help generate the JSON files automatically, the Vulkan SC samples support generating the JSON files with the Khronos JSON generation layer (VK_LAYER_KHRONOS_json_gen). However, VK_LAYER_KHRONOS_json_gen is a Vulkan layer, those binaries with names prefixed with vk are built using Vulkan API with this layer enabled. You can run those Vulkan samples on target to generate the JSON file automatically.

Host x86 Binary Build

Alternatively, you can generate the JSON files to run the Vulkan SC samples on the host. To do this, you must build the x86_64 binaries on the host as follows:

Prerequisites:

- > The desktop must have an NVIDIA dGPU.
- > Desktop Vulkan (version 1.2 or later) SDK and driver (from NVIDIA) must be installed.

Use these commands to build:

```
cd <NV_WORKSPACE>/drive-linux/samples/vulkan-sc-samples
mkdir build_host
cd build_host
cmake -DHOST_BUILD=ON ../
make
```

After the build, the following binaries are generated under the build_host/bin directory:

- > vk_01tri_host
- > vk_computeparticles_host

In addition, the data directory is also copied to the build_host/bin directory. The binaries and the data directory must be in the same directory.

6.1.7.6.2 Running the Vulkan SC Samples

This topic describes how to display Vulkan[®] SC information and how to run the Vulkan SC samples. OpenWFD is supported as a display module for the Vulkan SC samples.

Components

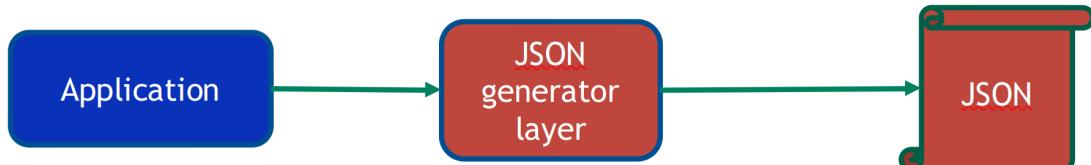
Vulkan SC applications can be developed from Vulkan applications. To develop a Vulkan SC application, you first create a Vulkan application with similar functionality and then extract the offline state needed for the Vulkan SC application. For running and developing Vulkan SC applications, various components are involved:

- > **Vulkan application:** The Vulkan application is used to develop the Vulkan SC application.
You use this application to extract the offline state needed for the Vulkan SC application.
- > **JSON generation layer:** This layer is used to generate the offline JSON pipeline state and the spir-v files that are needed for the PCC tool.
The library name is `libVkLayer_json_gen.so`.
- > **Pipeline Cache Compiler (PCC):** The PCC tool is used to generate the pipeline cache that the VKSC application loads.

Workflow

The following diagram outlines the workflow:

1. Generate JSON file during development



2. Generate pipeline cache container



Running the `vulkanscinfo` Sample (Target)

The `vulkanscinfo` command summarizes the core features of Vulkan SC API and saves the output to the `text/json/html` file.

- > To get a list of all available options for the `vulkanscinfo` command, specify the `--help` option:

```
cd <VulkanSC-Samples>/build/bin
./vulkanscinfo --help
```

- > To save the output to an HTML file, specify the `--html` option:

```
cd <VulkanSC-Samples>/build/bin
./vulkanscinfo --html
```

Running the 01tri Sample (Target)

1. Generate pipeline cache by running `vk_01tri`.

- a. Generate the JSON files.

The `VK_LAYER_KHRONOS_json_gen` layer is recommended to help automatically generate the JSON files. However, this layer is only supported on Vulkan; therefore, the Vulkan version of this `vk_01tri` sample with `VK_LAYER_KHRONOS_json_gen` enabled is also built.

Run `vk_01tri` on target to generate the JSON files as follows:

```
cd <VulkanSC-Samples>/build/bin
export VK_LAYER_PATH=/etc/vulkansc/icd.d
export VK_JSON_FILE_PATH=$PWD/data/pipeline/vksc_01tri/
./vk_01tri
```

Upon successful execution, the following files are generated in `VK_JSON_FILE_PATH`:

- > `vk_01tri_pipeline_0.json`
- > `vk_01tri_pipeline_0.vert.spv`
- > `vk_01tri_pipeline_0.frag.spv`

- b. Generate the pipeline cache by running the host PCC tool.

The PCC tool provided in the PDK is an `x86_64` binary; therefore, run the tool only on `x86_64` Linux host machine as follows:

```
cd <VulkanSC-Samples>/build/bin/data/pipeline/vksc_01tri
<NV_WORKSPACE>/drive-linux/vulkansc/pcc/Linux_x86-64/pcc -chip [gv11b|ga10b] -
path ./ -out pipeline_cache.bin
```

Upon successful execution, `pipeline_cache.bin` will be generated in the current folder.

2. Run the `vksc_01tri` sample.

The `vksc_01tri` sample already contains a default pipeline cache generated for `gv11b` and `ga10b` GPUs, and the cache binary in hex format is embedded in the `pipeline_cache.hpp` file that is built with the `vksc_01tri` binary. Therefore, you can run the sample directly by following these steps:

- a. Save the off-screen rendering result to an image file.

```
cd <VulkanSC-Samples>/build/bin
./vksc_01tri -o
```

After you run commands, an image file `tri.ppm` is generated. You can use any image viewer to open it.

- b. Display the rendering result by using OpenWFD.

```
cd <VulkanSC-Samples>/build/bin
./vksc_01tri -w
```

To run the vksc_01tri sample with the external pipeline cache generated in step 1.

```
cd <VulkanSC-Samples>/build/bin
./vksc_01tri -c -o
```



Note: By specifying the -c option, the application will look for the pipeline_cache.bin file in the ./data/pipeline/vksc_01tri directory. So pipeline_cache.bin generated in step 1 must be copied to that directory.

Running the computeparticles Sample (Target)

To read the texture data, the vksc_computeparticles sample uses the libktx.so external library that is located in <VulkanSC-Sample>/external/ktx/Linux_aarch64:\$LD_LIBRARY_PATH. Therefore, you must specify the path to the library before running the samples as follows:

```
export LD_LIBRARY_PATH=<VulkanSC-Sample>/external/ktx/Linux_aarch64:$LD_LIBRARY_PATH
```

1. Generate the pipeline cache by running vk_computeparticles.

- a. Generate the JSON files.

The VK_LAYER_KHRONOS_json_gen layer is **recommended** to help to automatically generate the JSON files. However, this layer is only supported on Vulkan, therefore the Vulkan version of this sample vk_computeparticles with VK_LAYER_KHRONOS_json_gen enabled is also built.

Run vk_computeparticles on target to generate the JSON files as follows:

```
cd <VulkanSC-Samples>/build/bin
export VK_LAYER_PATH=/etc/vulkansc/icd.d
export VK_JSON_FILE_PATH=$PWD/data/pipeline/computeparticles/
./vk_computeparticles
```

The following files are generated in VK_JSON_FILE_PATH:

- > vk_computeparticles_pipeline_0.json
- > vk_computeparticles_pipeline_0.frag.spv
- > vk_computeparticles_pipeline_0.vert.spv
- > vk_computeparticles_pipeline_1.json
- > vk_computeparticles_pipeline_1.compute.spv

This sample uses two different pipelines, graphics pipeline and compute pipeline; therefore, two JSON files are generated.

- a. Generate the pipeline cache by using the host PCC tool.

The PCC tool provided in the PDK is an x86_64 binary, therefore it can only be run on x86_64 Linux host machine as follows:

```
cd <VulkanSC-Samples>/build/bin/data/pipeline/computeparticles
<NV_WORKSPACE>/drive-linux/vulkansc/pcc/Linux_x86-64/pcc -chip [gv11b|ga10b] -
path ./ -out pipeline_cache.bin
```

Upon successful execution, pipeline_cache.bin will be generated in the current folder.

2. Running the vksc_computeparticles sample.

The `vksc_computeparticles` sample already contains a default pipeline cache generated for `gv11b` and `ga10b` GPUs, and the cache binary in hex format is embedded in the `pipeline_cache.hpp` file that is built with the `vksc_computeparticles` binary. Therefore, you can run the sample directly.

- Save the off-screen rendering result to an image file.

```
cd <VulkanSC-Samples>/build/bin
./vksc_computeparticles -o
```

- Display the rendering result by using OpenWFD.

```
cd <VulkanSC-Samples>/build/bin
./vksc_computeparticles -w
```

To run the `vksc_computeparticles` sample with the external pipeline cache generated in step 1.

```
cd <VulkanSC-Samples>/build/bin
./vksc_computeparticles -c -o
```



Note: By specifying the `-c` option, the application will look for the `pipeline_cache.bin` file in the `./data/pipeline/vksc_computeparticles` directory, so `pipeline_cache.bin` generated in step 1 must be copied to that directory.

Alternative Method to Generate JSON Files (Host)

Regarding generating the JSON files described in step 1.a (Generate the JSON files) , an alternative method can automatically generate the JSON files by using the `x86_64` binaries that can be run on `x86_64` Linux host machines. To use this method, follow the instructions to build the `x86_64` binaries as described in [Host x86 Binary Build](#).

Prerequisites:

- > Desktop Vulkan (version 1.2 or later) SDK and driver (from NVIDIA) must be installed.
- > The `VK_LAYER_KHRONOS_json_gen` library must be available on the host.

To run the `vk_01tri_host` sample on host,

```
cd <VulkanSC-Samples>/build_host/bin
export VK_LAYER_PATH=<NV_WORKSPACE>/drive-linux/vulkansc/ecosystem/vulkan-sc-layers/
json_generation_layers/binaries/Linux_x86_64
export VK_JSON_FILE_PATH=$PWD/
./vk_01tri_host
```

To run the `vk_computeparticles_host` sample on host,

```
cd <VulkanSC-Samples>/build_host/bin
export VK_LAYER_PATH=<NV_WORKSPACE>/drive-linux/vulkansc/ecosystem/vulkan-sc-layers/
json_generation_layers/binaries/Linux_x86_64
export VK_JSON_FILE_PATH=$PWD/
LD_LIBRARY_PATH=<VulkanSC-Sample>/external/ktx/Linux_x86_64 ./vk_computeparticles_host
```

6.1.8 Vulkan SC Guidance

This section provides guidance on how to use the validation layer as a development tool.

6.1.8.1 Validation Layer

To avoid generating faults, each application must comply with all **must** statements in the Vulkan SC Specification intended for users of the Vulkan SC API. This includes but is not limited to statements concerning:

- > Object lifetime
- > Threading behavior
- > Externally synchronized parameters
- > Host synchronization
- > Valid usage

To accomplish this compliance, developers should test their applications of Vulkan SC during development using the Khronos Vulkan validation layer for Vulkan SC, which NVIDIA has approved for use with this product. The name of the layer is `VK_LAYER_KHRONOS_validation`. This practice ensures that all API parameters follow the valid usage statements in the Vulkan SC Specification, and do not generate validation errors.

As rationale, the NVIDIA Vulkan SC library does not check all valid usage statements during operation. This is for reasons of run-time efficiency and to reduce code complexity. The `VK_LAYER_KHRONOS_validation` layer, when installed and enabled, logs violations of these valid usage statements via `VK_EXT_debug_utils` extension message callback. Developers should correct their API usage during development to prevent messages of error and warning severity. The Vulkan SC Specification at *Fault and Error Handling* also recommends use of validation layers during application development.

In an NVIDIA DRIVE® OS SDK installation, vksc-ecosystem packages distribute the Khronos Vulkan validation layer for use with Vulkan SC. The Platforms DRIVE® OS QNX Standard and DRIVE® OS Linux Standard support the validation layer. The vksc-ecosystem package is named with pattern `nv-driveos-<PLATFORM>-vksc-ecosystem-<VERSION>.deb`, where `<PLATFORM>` is either qnx or linux and `<VERSION>` is a version identifier for the SDK. The validation layer is provided as a source archive and also as a pre-built binary for aarch64 ISA. The vksc-ecosystem packages also distribute the loader library with file name `libvulkansc.so` and its source archive. The validation layer is deployed as a library with file name `libVkSCLayer_khronos_validation.so`. With the vksc-ecosystem package, the validation layer is installed on the target filesystem under `/usr/lib/` on Linux and its JSON manifest is installed to `/etc/vulkansc/icd.d/VkSCLayer_khronos_validation.json`. There is source archive and documentation at `<NV_WORKSPACE>/drive-<PLATFORM>/vulkansc/ecosystem/vulkan-sc-layers/validation_layers/`, where `<NV_WORKSPACE>` is local path to the DRIVE OS SDK installation, and `<PLATFORM>` is qnx or linux.

6.1.8.2 Fault Handling

This topic sets expectations to application developers on production use and behavior of Vulkan SC *Fault Handling*.

This Vulkan SC implementation expects that user applications monitor the Vulkan SC *Fault Handling* via `VkFaultCallbackInfo` and `vkGetFaultData`, as referenced below.

Avoid Generating Faults

User applications **must not** cause Vulkan SC to report faults at `VK_FAULT_LEVEL_CRITICAL` during normal operation. When applications correctly use the Vulkan SC API, and NVIDIA DRIVE® OS SEooC and NVIDIA DRIVE Orin™ SoC operate in normal conditions, this Vulkan SC implementation is designed to report **zero** faults via the *Fault Handling* interface introduced in the Khronos Vulkan SC Specification.

The rationale is that these reports are symptoms of detected faults. This Vulkan SC implementation of fault handling reports faults at only *Quality Managed (QM)* availability and automotive integrity level. Particularly for `VK_FAULT_LEVEL_CRITICAL`, the fault **can** indicate that the iGPU encountered an uncorrectable error.

To continue the rationale of why to avoid this error condition, the `VkDevice` that triggered the fault of `VK_FAULT_LEVEL_CRITICAL` becomes *lost*, as described in Vulkan SC Specification at 5.2.3. *Lost Device*. Vulkan SC API functions to submit or wait for `VkQueue` then return `VK_ERROR_DEVICE_LOST`. On Orin and QNX Safety Platform, the system's error response causes other `VkDevice` even in separate system processes to become *lost*, and also causes CUDA devices to return errors. This degrades the availability of iGPU execution for the system (Guest VM). An independent ASIL SW resource manager of NvGPU reports an asynchronous error to Safety Services, via the Error Propagation Library.

During development and verification, if the implementation reports faults to an application, application developers **should** investigate, and correct the usage of Vulkan SC API. To assist in logging and debugging, `VK_NV_private_vendor_info` extension supports a `VkFaultDataDescriptionNV` structure, which contains a description string.

Monitor SC Fault Handling

User applications **must** monitor the SC fault handling interface, for automotive deployment. The Vulkan SC Specification introduced a fault handling interface, at 35.1. *Fault Handling*. The Vulkan SC Specification identifies this interface as optional for applications, but this Vulkan SC implementation expects applications to monitor *Fault Handling*.

Specifically, this guide expects applications to exercise fault handling as below:

1. Application **must** register a callback function via `VkFaultCallbackInfo` when creating each `VkDevice`. Applications provide a function of signature `PFN_vkFaultCallbackFunction`, assign its function pointer to `VkFaultCallbackInfo::pfnFaultCallback`, and “pNext-chain” `VkFaultCallbackInfo` to `VkDeviceCreateInfo` passed to `vkCreateDevice`.

2. Application **must** call `vkGetFaultData` for each `VkDevice`, after completing all its calls to functions in *API Group: Init*.
3. After the application calls `vkQueueSubmit`, the application **should** call `vkGetFaultData`, within one second, on the `VkDevice` from which that `VkQueue` was received. The one-second interval is arbitrary. One call to `vkGetFaultData` can query any faults after multiple prior queue submissions.

As rationale, many functions in Vulkan SC API return `void`, rather than `VkResult` return code. This interface design supports efficient execution on CCPLEX with low count of branches, for application code-paths that execute frequently, for example to record command buffers. If the Vulkan SC implementation detects faults, and the functions, which return `void` do not execute normally, the SC fault handling reports to application that fault(s) occurred, at best effort.

Recoverable Faults

In production, applications **should not** generate faults even at the lower criticality `VkFaultLevel`. Those lower levels are `VK_FAULT_LEVEL_RECOVERABLE`, `VK_FAULT_LEVEL_WARNING`, and `VK_FAULT_LEVEL_UNASSIGNED`. This minimizes the accumulation of errors, and prevents multi-point failures, where the cause of recoverable fault is the first point, and the fault report and handling are additional points.

This guide suggests fault handling by applications for different faults of `VK_FAULT_LEVEL_RECOVERABLE`:

- When any `vkCmd`-prefixed function causes a fault during command recording, of `VK_FAULT_TYPE_COMMAND_BUFFER_FULL` or `VK_FAULT_TYPE_INVALID_API_USAGE`, Developers can expect `vkEndCommandBuffer` to then return an error `VkResult`. Applications can clear the error state of the `VkCommandBuffer` with `vkResetCommandPool` on its pool. Until that call to `vkResetCommandPool`, subsequent `vkCmd`-prefixed functions to record to that `VkCommandBuffer` will be silently ignored.
- For functions not prefixed with `vkCmd`, when those functions cause a fault of `VK_FAULT_TYPE_INVALID_API_USAGE`, the API function skips its nominal behavior. Developers can expect that retry with the exact same parameters will repeat the fault report. Applications can retry that function, or skip that call and enter alternate behavior.

6.2 EGL Interoperability and EGLStream

EGL is a [Khronos](#) defined API consisting of a core specification and optional extensions that provides an interface between APIs, as well as a connection to the window system or other underlying platform. It provides mechanisms for creating surfaces that client APIs can read and write, importing and exporting resources created by clients, and mapping window system resources into clients. This allows it to serve as a centralized interop hub to exchange shared resources, removing the need for specialized interops between individual clients.

EGLStream is an [EGL API extension](#) that provides a mechanism that efficiently transfers a sequence of image frames from one API to another. EGLStream is supported on the

following NVIDIA supported APIs: OpenGL ES, CUDA, and NvMedia, allowing frames to be shared between these APIs.

For additional details on EGL interoperability and EGLStream support, please consult the relevant documentation as follows:

- > [OpenGL ES](#)
- > [NvMedia](#)
- > [CUDA Toolkit](#)

6.3 Window Systems

This topic describes the three window systems for Linux:

- > X11
- > Wayland
- > EGLDevice/EGLOutput

The following table lists the window system name and its main library.

Window System	Main Library
X11 Window System	X Server
Wayland Window System	Weston Compositor
EGLDevice/EGLOutput	No main library.

6.3.1 Wayland Window System

Wayland is a protocol that a backend compositor uses to communicate with its clients. It is also a C library implementation of that protocol. Weston is the reference implementation of the Wayland compositor. The platform supports Wayland and Weston. Check the Release Notes for specific versions supported.

For more information about Wayland, see the Wayland home page and documentation page at:

<http://wayland.freedesktop.org/>
<http://wayland.freedesktop.org/docs/html/>

6.3.1.1 EGLOutput/EGLDevice Specifications

NVIDIA defines extensions to enumerate and control devices and screens through EGL. These are collectively referred to as EGLOutput/EGLDevice.

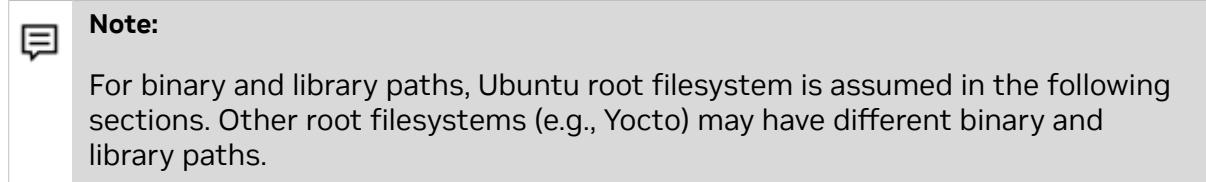
The EGLOutput/EGLDevice specifications used by Weston are as follows:

Extension	Link
EGL_EXT_device_base—Allows the initialization of EGL displays directly on top of native GPU or device objects.	https://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_device_base.txt
EGL_EXT_device_drm—Allows mapping of device between EGL and DRM/KMS.	https://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_device_drm.txt
EGL_EXT_output_base—Allows rendering to be directed to a screens or display outputs.	https://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_output_base.txt
EGL_EXT_output_drm— Allows mapping of output handles between EGL and DRM/KMS.	https://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_device_drm.txt
EGL_EXT_stream_consumer_egloffput— Allows the binding of EGLOutputLayerEXTs as stream consumers.	https://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_stream_consumer_egloffput.txt
EGL_KHR_stream_consumer_gltexutre — Allows an OpenGL(ES) texture to be connected to an EGLStream as its consumer.	https://www.khronos.org/registry/egl/extensions/KHR/EGL_KHR_stream_consumer_gltexutre.txt

The Weston compositor uses EGLOutput/EGLDevice to display the composited Weston desktop or individual Wayland applications on a physical display device.

6.3.1.2 Runtime Configuration

The platform provides utilities for changing the configuration and runtime parameters.



The full source code to the NVIDIA implementation of Weston is in:

```
drive-linux/samples/wayland/weston
```

The NVIDIA implementation of Weston uses the Direct Rendering Manager (DRM) backend, found in the target filesystem at:

```
/usr/local/lib/weston/drm-backend.so
```

There are two implementations of the DRM display backend. One is implemented on the Linux KMS. The other is implemented on top of the EGLOutput series of extensions.

During the Weston startup, Weston creates an OpenGL ES EGLStream producer, if option use-egldevice is enabled. Weston then creates EGLOutput. EGLOutput is the EGLStream consumer that passes all the output frames from Weston to the display. For more information, see [EGLOutput Specifications](#). If use-egldevice is not enabled, the output method switches to drm/kms/Wayland buffer sharing between EGL clients and Weston is implemented using EGLStreams. The EGL client creates an EGLStream producer and binds the EGLStream file descriptor to wl_buffer through the Wayland protocol. Weston queries the EGLStream file descriptor from wl_buffer through a query of EGL_WAYLAND_BUFFER_WL to eglQueryWaylandBufferWL(). This functionality is provided by the EGL_WL_bind_wayland_display extension.

Wayland buffer sharing also supports dma-buf sharing based on the linux-dmabuf-unstable-v1-protocol protocol. For more information, see [Weston dma-buf Support](#).

The NVIDIA EGL implementation tries to detect what platform it is running on at runtime (e.g., Wayland, etc.). It is possible to bypass this detection by setting the EGL_PLATFORM environment variable. For Wayland, this variable must be set to "wayland". And do not set the environment variable DISPLAY in case of the Wayland platform. If DISPLAY is set, you are assumed to be running X11.

6.3.1.3 libdrm Support

The libdrm.so library is used to set display modes and to attach framebuffer images to display overlays. The platform custom implementation of libdrm.so is not implemented on top of the DRM-KMS display driver. Instead, it is implemented on top of the NVIDIA NVDC display driver.

Weston requires the NVIDIA custom libdrm.so and is not compatible with the default Ubuntu libdrm.so.

The default Ubuntu libdrm libraries are located at:

```
/usr/lib/aarch64-linux-gnu/libdrm.so*
```

The SDK installation deletes those libraries and replaces them with libdrm.so at:

```
/usr/lib/libdrm.so
```

When installing 3rd party packages or new software with apt-get, you must make sure that libdrm.so* shared objects are not recreated. If they exist, remove them from:

```
/usr/lib/aarch64-linux-gnu/
```

6.3.1.4 DRM backend

6.3.1.4.1 Supported renderers

- > WESTON_DRM_BACKEND_RENDERER_PIXMAN
 - Software rendering in composition.
- > WESTON_DRM_BACKEND_RENDERER_HAL
 - Use VIC and GPU in composition. It is not available on automotive platforms.
- > WESTON_DRM_BACKEND_RENDERER_GL
 - Use EGL/OpenGL in composition.

6.3.1.4.2 Supported Output methods

- > DRM_OUTPUT_METHOD_EGLSTREAM
 - Use EGLOutput. EGLOutput is the EGLStream consumer that passes all the output frames from Weston to the display. For more information, see [EGLOutput Specifications](#).
 - DRM_OUTPUT_METHOD_GBM_SURFACE
 - Default output method. If not enabled, use-egldevice.
 - DRM_OUTPUT_METHOD_SWAPCHAIN
 - Internal maintained framebuffer swapchain if EGL does not support platform_gbm.

6.3.1.5 Weston Common Options

Weston supports multiple backends. This section describes the core options that all Weston backends support.

Command	Parameter(s)	Description
--version	N/A	Prints the Weston version.
-B, --backend	drm-backend.so eglstream-backend.so (for cross partition use cases)	Specifies the backend module. Defaults to <code>drm-backend.so</code> . Usage: <code>--backend=<module></code> , where <code>module</code> is one of the parameters.
--shell	desktop-shell.so ivi-shell.so	Specifies the shell module. Defaults to <code>desktop-shell.so</code> . Usage: <code>--shell=<module></code> , where <code>module</code> is one of the parameters.

Command	Parameter(s)	Description
-S, --socket	<socket name>	<p>Listens to the specified socket name.</p> <p>Usage: --socket=<socket name>for applications.</p>
-i, --idle-time	<seconds>	Specifies the idle time in seconds.
--log	<file name>	<p>Displays the log for the given file.</p> <p>Usage: --log=<file name></p>
-h, --help	N/A	Displays the help message.

6.3.1.6 Weston Backend Options

Weston supports multiple backends and, depending on the backend used, accepts different options. NVIDIA supports `drm-backend.so`, which allows Weston to support NVIDIA graphics.

6.3.1.6.1 `drm-backend.so` options

The supported `drm-backend.so` options and the values they consume are as follows.

Command	Parameter(s)	Description
--seat	<SEAT>	The seat that Weston should run on, instead of the seat defined in XDG_SEAT
--tty	<TTY>	Select which tty to use
--drm-device	<CARD>	The DRM device to use, e.g. "card0".
--use-pixman	N/A	Use the pixman (CPU) renderer

Command	Parameter(s)	Description
--use-hal	N/A	Use the HAL renderer
--use-egldevice	N/A	Use the EGLDevice and EGLOutput with GL render
--current-mode	N/A	Specifies that current DRM mode is preferred over EDID mode. If the display is already turned on by any DRM application other than Weston, Weston uses the current DRM mode instead of a new display mode from the monitor EDID.

6.3.1.7 Weston Configuration File Location

Weston uses a configuration file called `weston.ini` for its setup. When Weston starts, it will search for the `weston.ini` configuration file in one of the following places:

If `$XDG_CONFIG_HOME` is set, Weston searches in:

```
$XDG_CONFIG_HOME/weston.ini
```

If `$HOME` is set, Weston searches in:

```
$HOME/.config/weston.ini
```

If `$XDG_CONFIG_DIRS` is set, Weston searches in:

```
$XDG_CONFIG_DIR/weston/weston.ini
```

If `$XDG_CONFIG_DIRS` is not set, Weston searches in:

```
/etc/xdg/weston/weston.ini
```

If no variables are set, Weston searches in:

```
<current dir>/weston.ini
```

Where the environment variable:

- `$HOME` is your home directory.
- `$XDG_CONFIG_HOME` is your specific configuration directory.
- `$XDG_CONFIG_DIRS` is a colon ':' delimited list of configuration base directories, such as `/etc/xdg-foo:/etc/xdg`.

6.3.1.8 Weston Display Configuration

Weston picks the default settings for display by itself. If the default settings are insufficient, the display settings can be defined in `weston.ini`.

To configure the display, add the following to `weston.ini` with the appropriate values for output name, mode, and transform:

```
[output]
name=<output_name>
mode=<mode>
transform=<transformation>
```

Where `name=<output_name>` sets a name for the output string. The backend uses the name to identify the output.

Output Name	Description
LVDS1	Laptop internal panel number 1.
VGA1	VGA connector number.

The `mode=<mode>` assignment sets the output mode string. The mode parameter is handled differently depending on the backend. The DRM backend accepts different modes:

Mode	Description
WIDTHxHEIGHT	Resolution size width and height in pixels.
preferred	Uses the preferred mode.
current	Uses the current CRT controller mode.
off	Disables the output.

Optionally, you can specify a modeline, for example:

```
173.00 1920 2048 2248 2576 1080 1083 1088 1120 -hsync +vsync
```

A modeline consists of the refresh rate in Hz, the horizontal and vertical resolution, and options for horizontal and vertical synchronization. The `cvt(1)` program provides a suitable modeline string.

The `transform=<transformation>` assignment applies the transformation to the screen output string. The transform key must be one of the following 8 strings:

Transform string	Description
normal	Normal output.
90	90 degrees clockwise.
180	Upside down.
270	90 degrees counter clockwise.
flipped	Horizontally flipped.
flipped-90	Flipped and rotated 90 degrees clockwise.
flipped-180	Flipped upside down.
flipped-270	Flipped and 90 degrees counter clockwise.

The following is a `weston.ini` configuration example:

```
[output]
name=LVDS1
mode=1680x1050
transform=90
icc_profile=/usr/share/color/icc/colord/Bluish.icc
```

6.3.1.9 Prerequisites to Starting Weston

Weston creates its Unix socket file, for example `wayland-0`, in the directory specified by the required environment variable `$XDG_RUNTIME_DIR`. `$XDG_RUNTIME_DIR` defines the base directory relative to which user-specific non-essential runtime files and other file objects (such as sockets, named pipes, etc.) are stored.

Check whether the `$XDG_RUNTIME_DIR` environment variable is set, if it is not set please set it as follows.

6.3.1.9.1 Setting the `$XDG_RUNTIME_DIR` Directory

Use either of the following two methods to set the `$XDG_RUNTIME_DIR` directory before running Weston:

- > Set `$XDG_RUNTIME_DIR` manually
- > Set `$XDG_RUNTIME_DIR` using your shell profile

6.3.1.9.2 To set the `$XDG_RUNTIME_DIR` directory manually

- > Execute the following:

```
mkdir /tmp/xdgruntime
```

```
chmod 700 /tmp/xdgruntime
export XDG_RUNTIME_DIR=/tmp/xdgruntime
```

At runtime, `/tmp/xdgruntime` looks like:

```
root@nvidia:/# ls -lt /tmp/xdgruntime
total 0
srwxr-xr-x 1 root root 0 May 12 11:31 wayland-0
-rw-r----- 1 root root 0 May 12 11:31 wayland-0.lock
```

6.3.1.9.3 To set the `$XDG_RUNTIME_DIR` directory using your shell profile

- > Modify your shell profile to perform the manual steps described above. For information on modifying your shell, see the manual for your shell. Also see these important tips for when modifying your shell profile to set up `$XDG_RUNTIME_DIR`:
 - Determine the correct shell profile to modify.

Some shells read multiple files (Zshell). Others pick the first available file, ignoring the others (Bash). `~/.profile` is generally a good guess for most Bourne-shell compatible shells; `~/.zprofile` is the Zshell equivalent. Use the profile file if there is one; otherwise, use the login file. The software automatically uses the variable once it is set. This is useful if you want to use your profile file on different systems.

- Put the following code in your shell profile and adapt it to your shell's internals. This code is Bourne-shell compatible.

```
if test -z "${XDG_RUNTIME_DIR}"; then export
XDG_RUNTIME_DIR=/tmp/${UID}-runtime-dir
if ! test -d "${XDG_RUNTIME_DIR}"; then mkdir
"${XDG_RUNTIME_DIR}"
chmod 0700 "${XDG_RUNTIME_DIR}"
```

6.3.1.10 Starting Weston

Weston can be launched with root privileges or via `weston-launch` tool.

6.3.1.10.1 To start Weston with root privileges

- > Execute the following command to switch to superuser:

```
sudo su
```

The default password is `nvidia`.

- > Create symbol links:

```
sudo ln -sf /usr/lib/libnvgbm.so /usr/lib/aarch64-
linux-gnu/libgbm.so.1
sudo ln -sf /usr/lib/libdrm.so.2 /usr/lib/aarch64-
linux-gnu/libdrm.so.2
```

- > Load Weston:

```
unset DISPLAY
mkdir /tmp/xdg
chmod 700 /tmp/xdg
```

```
export XDG_RUNTIME_DIR=/tmp/xdg
export WESTON_TTY=1
sudo insmod /lib/modules/$(uname -r)/extra/opensrc-disp/nvidia-drm.ko modeset=1
sudo XDG_RUNTIME_DIR=/tmp/xdg weston --tty="$WESTON_TTY" --idle-time=0 &
```



Note: If you want to use the X11 environment, remove nvidia-drm driver (rmmod nvidia_drm) or restart the board.

- > Launch Weston as specified below.

6.3.1.10.2 To start Weston on desktop-shell

- > Execute the following command:

```
weston --idle-time=0 &
```

By default, Weston runs using desktop-shell.

6.3.1.10.3 To start Weston with ivi-shell and hmi-controller

- > Execute the following command:

```
weston --tty="$WESTON_TTY" --idle-time=0 --shell=ivi-shell.so --modules=hmi-
controller.so &
```

ivi-shell can pull different controllers that effectively take care of the window management.



Note:

By default, weston.ini sets the background-color to 0xff000000 (ARGB color). Therefore, a plain, dark background is expected.

6.3.1.10.4 To start Weston with ivi-shell and ivi-controller

```
weston --shell=ivi-shell.so --modules=ivi-controller.so
```

6.3.1.10.5 To start Weston with ivi-shell, ivi-controller and ivi-input-controller.so

- > Modify weston.ini to load ivi-input-controller:

```
[ivi-shell]
ivi-input-module=ivi-controller.so
```

- > Execute the following command:

```
weston --shell=ivi-shell.so &
```

This starts Weston with IVI shell using the corresponding controller module. The display should remain dark. The ivi-controller does not provide any desktop decorations, by default.

6.3.1.10.6 To start Weston without root privileges

Weston can be launched as non-root with the weston-launch binary. It is present in /usr/local/bin for Ubuntu rootfs. Follow the steps below to launch Weston as non-root.

1. Add the non root user to weston-launch group:

```
sudo su
usermod -a -G weston-launch <non_root_user_name>
chown root /usr/local/bin/weston-launch
chmod +s /usr/local/bin/weston-launch
```

2. Launch Weston with weston-launch binary as non-root:

```
su <non_root_user_name>
weston-launch [args...] [-- [weston args..]]
```

For example:

- > Run weston-launch with desktop-shell:
weston-launch -- --shell=desktop-shell.so
- > Run weston-launch with ivi-shell:
weston-launch -- --shell=ivi-shell.so

6.3.1.11 Running Weston Samples

The instructions provided assume Weston and Wayland are running as superuser.

6.3.1.11.1 To start Weston-simple-egl with ivi-shell and hmi-controller

- > Execute the command:

```
weston-simple-egl &
```

6.3.1.11.2 To start Weston-simple-egl with ivi-shell and ivi-controller

1. Execute the command:

```
weston-simple-egl -width 1920 -height 1080 -surface 10 &
```

2. Manually configuring the client surface:

```
LayerManagerControl set surface 10 source region 0 0 1920 1080
LayerManagerControl set surface 10 destination region 0 0 1920 1080
LayerManagerControl set surface 10 visibility 1
```

Nothing appears on the screen because the application surface is not linked to a layer.

3. Create a layer and attach the client surface to it:

```
LayerManagerControl create layer 1000 1920 1080
LayerManagerControl set layer 1000 render order 10
LayerManagerControl set layer 1000 visibility 1
```

4. Link your layer to a screen:

```
LayerManagerControl set screen 0 render order 1000
```

6.3.1.12 Compositing Mode in Weston

Weston supports mixed mode compositing using dma-buf. Weston collects all the overlay planes and assigns one to the GFX as the primary plane. Others are available as overlay planes. Weston evaluates all surfaces and views on every frame, and chooses one of these strategies to composite them:

- DRM_OUTPUT_PROPOSE_STATE_MIXED (Overlay + GL)
- DRM_OUTPUT_PROPOSE_STATE_RENDERER_ONLY (GL)
- DRM_OUTPUT_PROPOSE_STATE_PLANES_ONLY (Overlay only)

The weston-simple-dmabuf-egldevice demo application demonstrates mixed-mode compositing path in weston, where some surfaces are assigned overlay planes and others are composited with GL.

6.3.1.12.1 Display Hardware Compositing in Weston

The DRM compositor in Weston calls drmModeAddFB2WithModifiers() to associate a DRM framebuffer ID with the dma-buf it receives via Linux DMA-BUF Unstable V1 Protocol. It then uses the framebuffer ID to present the dma-buf to one of the available display hardware planes using drmModeSetPlane(), in the legacy path, or drmModeAtomicCommit(), in the atomic modeset path. This takes the form of a flip, where the reference to the new buffer is swapped in during vblank.

6.3.1.13 weston-debug

The weston-debug client application receives and prints debug messages using the weston-debug protocol. For example, when Weston is recompositing, it prints all surface views and descriptions of the decision process regarding their assignment to overlays and GL compositing.

For weston-debug to work, you must run Weston with the --debug switch. Do not use this switch in production.

You can pass weston-debug additional parameters to choose categories of debug output to display:

```
$ sudo XDG_RUNTIME_DIR=/tmp/xdg weston --debug
$ sudo XDG_RUNTIME_DIR=/tmp/xdg ./weston-debug -a
```

Alternatively, you can enable Weston debug output by setting the environment variable WAYLAND_DEBUG=server, and Weston client debug by setting WAYLAND_DEBUG=1.

6.3.1.14 Weston dma-buf Support

Clients can post dma-buf buffers to Weston using Wayland's Linux DMA-BUF Unstable V1 Protocol. Dma-buf support with Weston contains the following components.

6.3.1.14.1 Buffer Allocation

GBM is commonly used to allocate buffers, which are backed by the dma-buf file descriptor.

This code snippet shows how to allocate a buffer using GBM:

```
drm_fd = open("/dev/dri/card0", O_RDWR);
device = gbm_create_device(drm_fd);
bo = gbm_bo_create_with_modifiers(device, width, height,
                                    format, modifiers, modifiers_count);
```

Weston supports the DRM_FORMAT_XRGB8888 and DRM_FORMAT_ARGB8888 formats.

A **modifier** is an encoding of vendor-specific buffer layout parameters. Weston modifiers are supported by DRM (read the IN_FORMATS plane property blob) and by EGL (call `eglQueryDmaBufModifiersEXT()`).

6.3.1.14.2 Buffer Read/Write from CPU

You can use this API function to write data directly into a GBM buffer:

```
gbm_bo_write(bo, user_buffer, sizeof user_buffer);
```

You can also memory map a GBM buffer and get a CPU-accessible address to the data written to or read from it:

```
uint32_t dst_stride = 0;
void *gbo_mapping = NULL;
// Map bo to CPU accessible address.
char *dst_ptr = gbm_bo_map(bo,
                           0, 0,
                           width,
                           height,
                           GBM_BO_TRANSFER_READ_WRITE,
                           &dst_stride,
                           &gbo_mapping);
// Write data to or read data from dst_ptr.
. . .
// Unmap bo.
gbm_bo_unmap(bo, gbo_mapping);
```

6.3.1.14.3 Wayland Protocol to Post dma-buf Buffers to Weston

Weston uses the Generic Buffer Management (GBM) library to allocate buffers that are backed by dma-buf file descriptors. Client applications must call the following GBM functions to get the dma-buf file descriptor and associated parameters from the GBM buffer object. To post the dma-buf to a Wayland surface, the file descriptor and parameters must be provided via Linux DMA-BUF Unstable V1 Protocol.

The following code snippet shows the main calls involved (assume gbm buffer object contains one plane only).

```
stride = gbm_bo_get_stride_for_plane(bo, 0);
offset = gbm_bo_get_offset(bo, 0);
```

```

handle = gbm_bo_get_handle_for_plane(bo, 0);
modifier = gbm_bo_get_modifier(bo);
drmPrimeHandleToFD(drm_fd, handle, 0, &dmabuf_fd);
zwp_linux_buffer_params_v1_add(params, dmabuf_fds, 0, offset,
                                stride, modifier >> 32,
                                modifier & 0xffffffff);
zwp_linux_buffer_params_v1_create(params, width, height,
                                  format, flags);

```

6.3.1.14.4 GL Renderer in Weston

The GL renderer in Weston creates an `eglImage` from a `dma-buf` object that it receives with Linux DMA-BUF Unstable V1 Protocol using the `EGL_EXT_image_dma_buf_import` and `EGL_EXT_image_dma_buf_import_modifiers` extensions. Weston obtains the GL texture target to be used with this `eglImage` by calling `eglQueryDmaBufModifiersEXT()`, and binds the `eglImage` to the texture.

6.3.1.15 Gnome-Wayland Desktop Shell Support

To enable experimental Gnome-Wayland desktop shell support:

1. Install `gdm3`, `mutter`, and their dependencies:

```

sudo apt update
sudo apt install -y gdm3 mutter adwaita-icon-theme-full
sudo apt install -y --reinstall libdrm2 ubuntu-session

```

2. Repair the broken `drm-nvdc` soft link and rename the Ubuntu session file:

```

sudo ln -sf /usr/lib/libdrm.so.2 /usr/lib/aarch64-
linux-gnu/libdrm.so.2
sudo mv /usr/share/wayland-sessions/ubuntu-wayland.desktop
/usr/share/wayland-sessions/ubuntu.desktop

```

3. Add the `gdm` daemon to the video group:

```
sudo usermod -a -G video gdm
```

4. Configure `gdm` to use Wayland by setting the following flag in the `[daemon]` section of `/etc/gdm3/custom.conf`:

```

[daemon]
WaylandEnable=true

```

5. Load the `tegra-udrm` kernel module:

```
sudo modprobe tegra-udrm modeset=1
```

6. Start `gdm`:

```
sudo systemctl start gdm3.service
```

7. Log in and verify the Wayland backend is running:

```
ps -e | grep wayland
```

8. Configure the `tegra-udrm` kernel module to load on boot:

```

sudo bash -c 'echo "options tegra-udrm modeset=1" > /lib/modprobe.d/tegra-udrm.conf'
sudo bash -c 'echo "tegra-udrm" >> /etc/modules-load.d/modules.conf'

```

9. Reboot and repeat step 7 to verify the Wayland backend auto-started successfully.

6.3.2 EGLDevice

This topic describes EGL mechanisms that you can use to implement a pure EGL display. Such a display does not use a window system.

EGLDevice

EGLDevice provides a mechanism to access graphics functionality in the absence of or without reference to a native window system. It is a method to initialize EGL displays and surfaces directly on top of GPUs/devices rather than native window system objects. It is a cross-platform method to discover media devices like displays, GPUs, etc. The set of EGLDevice extensions boot strap EGL, without the use of any native APIs.

EGLOutput

EGLOutput is to graphical outputs what EGLDevice is to devices. It allows enumeration of outputs on a device. EGLOutput allows rendering directly to a screen in the absence of a window system. Additionally, it allows applications to bypass native window systems for direct rendering. It defines certain EGL resources for referencing display control hardware associated with an EGL device. EGLOutput provides a binding between GL, NVIDIA® CUDA®, and multimedia rendering and the display output. In a typical Embedded setting, the outputs are often initialized to a fixed state at system startup. But in cases where they are configurable, other interfaces such as DRM must be used.

EGLStream

EGLStream is a mechanism to share data efficiently between different APIs without copying data. APIs could be OpenGL, CUDA, Multimedia, etc. A producer and a consumer are attached to two ends of a stream object:

- Producer adds content into the stream.
- Consumer retrieves this content.

EGLOutput instances can also be specified as consumers, allowing APIs to direct their output to the screen.

6.3.2.1 Running EGLDevice Samples

This section describes the instructions to run EGLDevice samples.



Note: Ensure that no other Graphic samples are running on the target.

1. Load the following driver modules:

```
sudo insmod /lib/modules/$(uname -r)/extra/opensrc-disp/nvidia.ko
rm_firmware_active="all"
sudo insmod /lib/modules/$(uname -r)/extra/opensrc-disp/nvidia-modeset.ko
sudo insmod /lib/modules/$(uname -r)/extra/opensrc-disp/nvidia-drm.ko modeset=1
```

2. Change to the sample directory:

```
cd /opt/nvidia/drive-linux/samples/opengles2/bubble/egldevice
```

3. Start the sample as the root user.

```
#./bubble -windowsize 1920 1080 -sec 15
```



Note: This step only applies to OOBE-RFS filesystems.

These instructions are applicable to other EGLdevice samples. For samples binaries, compile the samples in the host using the `make` command and copy the entire sample directory into the target.

If you want to run X11 samples, remove the `nvidia_drm` module (`#rmmod nvidia_drm`) and see [To start the X server](#).

6.3.2.2 Extensions

EGL extensions specify the behavior, procedures, and functions for these EGL mechanisms.

- > EGLDevice
 - EGL_EXT_device_base
 - EGL_EXT_platform_base
 - EGL_EXT_platform_device
- > EGLOutput
 - EGL_EXT_output_base
- > EGLStream
 - EGL_KHR_stream
 - EGL_KHR_stream_producer_eglsurface
 - EGL_EXT_stream_consumer_egloutput

For a description of these EGL extensions, see:

- > [EGL and EGL Extensions](#)
- > Khronos EGL extension specifications at:

<https://www.khronos.org/registry/egl>

6.3.2.3 Runtime Configuration

6.3.2.3.1 Conditions Requiring a Stream Surface

In the absence of an underlying window system, the stream surface is necessary for on screen rendering. Stream surfaces behave like any other EGLSurface. For example, the `eglSwapBuffers` function must still be called to indicate the end of a frame. However, stream surfaces are stream producers, so `eglSwapBuffers` submits rendering to the stream rather than presenting it to a native window directly. Typically, an EGLOutput consumer can use any producer attached to the stream but needs a surface producer when rendering API is OpenGL. For more information, see [Creating a Stream Surface](#) in this topic.

A common use case for stream producer surfaces is an application producing display frames using OpenGL, attached as a producer to one end of an EGL stream. The stream consumer on the other end is an EGLOutput layer sending frames directly to a display device.

6.3.2.4 Implementation

The following two sub-sections describe the steps to render to an EGL device using a stream surface.

6.3.2.5 Rendering to EGLDevice

These are the steps to render to an EGL device (more detailed steps follow):

1. Create an EGL display from an EGL device.
2. Create an EGL context from the EGL display.
3. Create an EGL stream producer surface.
4. Bind a GL context to the stream surface, i.e. make current.
5. Post the surface contents to the stream using swap buffers.

6.3.2.6 Creating a Stream Surface

Creating a stream surface uses the following functions:

Function	Extension and Function Description
eglCreateStreamProducerSurfaceKHR	EGL_KHR_stream_producer_eglsurface Creates an EGLSurface and connects it as the producer of a stream.
eglStreamConsumerOutputEXT	EGL_EXT_stream_consumer_egloutput Binds an EGLOutput layer as a stream consumer to send rendering directly to a display device.

According to the EGL_KHR_stream specification, the EGLStream cannot be used until it has been connected to a consumer and producer. The consumer must be connected before the producer is connected.

6.3.2.6.1 To render to an EGLDevice through stream

1. Query EGL extensions using `eglGetProcAddress`.
2. Query available EGLDevices with `eglQueryDevicesEXT`.
3. Obtain an EGL display from the EGL device using `eglGetPlatformDisplayEXT`.

This step creates an EGLDisplay that does not belong to any native platform.

4. Initialize/setup EGL using `eglInitialize`.
5. Setup an EGLOutput. For detailed steps, see [Setting Up the Display with OpenWFD and EGL Device](#).
 - > Selecting an output:
 - Can be done by enumerating all outputs and selecting a known index.
 - Can be done by looking up an output associated with a native (e.g. DRM) screen handle.
 - > If necessary, initialize display settings using native interfaces
6. Direct rendering to an EGLOutput.
 - > Create an EGL stream using `eglCreateStreamKHR`.
 - > Connect the output layer to the stream. Bind consumer end of stream to EGLOutput window object using `eglStreamConsumerOutputEXT`.
7. Set buffer configurations by choosing an EGLConfig
8. Create a stream producer surface to feed the stream using `eglCreateStreamProducerSurfaceKHR`.
9. Create an EGL context, make it current by binding it to the stream surface using `eglMakeCurrent`.
10. Post surface contents to the stream using `eglSwapBuffers`.

6.3.2.6.2 To use an EGLStream in cross-process mode

1. Make the following additions and changes to `nvm_egistream.int`:

```

AddressSpace nvm_egistream_producer
  Filename      nvm_egistream_as0
  Arguments     -producer 0 -f /nfsmount/welcome_animation.264 -standalone 1
  MemoryPoolSize 32M
  ExtendedMemoryPoolSize 64M
  HeapExtensionReservedSize 64M
  Language      C
  Task          Initial
    StartIt      false
    StackLength   2M
  EndTask
EndAddressSpace
AddressSpace nvm_egistream_consumer
  Filename      nvm_egistream_as0
  Arguments     -consumer 0 -d 2 -standalone 2
  MemoryPoolSize 32M
  ExtendedMemoryPoolSize 64M
  HeapExtensionReservedSize 64M
  Language      C
  Task          Initial
    StartIt      false
    StackLength   2M
  EndTask
EndAddressSpace

```

2. Rebuild the application.

3. Load the application binary with the following command:

```
Load /nfsmount/nvm_eglstream_egldevice
```

4. Start the consumer with the following command:

```
rt nvm_eglstream_consumer Initial
```

5. Start the producer with the following command:

```
rt nvm_eglstream_producer Initial
```

This procedure enables a video producer and video consumer combination. The same procedure can also be extended for other producer/consumer combinations.

6.3.2.7 Cross-Process and Cross-Partition EGLStream Applications

This release includes the simple cross-process EGLDevice consumer and producer applications `helloconsumer` and `hellopublisher`. The applications listen on 127.0.0.1, port 8888 by default. They can be used as cross-partition applications by passing `--crosspart` or `-c` to the consumer and `--crosspart [<consumer_ip>]` or `-c [<consumer_ip>]` to the producer.

The applications are built into `demo_dd`.

6.3.2.7.1 To use the applications

1. Load `demo_dd`.

2. Start `helloconsumer` on the target with the following command:

```
rt helloconsumer Initial
```

3. Start `hellopublisher` on the target with the following command:

```
rt hellopublisher Initial
```

If the applications run correctly, the upper-left corner of display 2 contains a flashing square.

Using `EGL_KHR_stream_consumer_gltexure` functionality, the EGLStream can also be bound to a gltexture consumer (requiring an EGLContext.) The consumer texture buffer can then be rendered to an EGLSurface, for example.

For more information on `EGL_KHR_stream_consumer_gltexure` see the following website: https://www.khronos.org/registry/EGL/extensions/KHR/EGL_KHR_stream_consumer_gltexure.txt

6.3.2.8 Connecting a Surface to a Screen

You must use an EGLStream to connect a surface (i.e., a stream surface) to a screen. The surface is the stream producer, and the screen is the stream consumer.

- > `eglGetPlatformDisplayEXT` returns an EGLDisplay handle belonging to a screen as specified by platform argument in the function call.
- > `eglMakeCurrent` attaches an EGL rendering context to draw and read an EGL surface. It also binds a context and a surface to the current rendering thread.

- > Steps 6 through 9 in [Creating a Stream Surface](#) are required for connecting a stream surface to a screen.
- > `eglGetPlatformDisplayEXT` returns a handle to screen.
- > `eglMakeCurrent` binds the handle to the current context and surface.

6.3.2.9 Setting Up the Display with OpenWFD and EGL Device

OpenWFD (Open Windowing Foundation – Display) is a Khronos specification that provides a low-level hardware abstraction interface to use the display hardware. OpenWFD can be used with an EGL device to interact with the display hardware.

Selecting a WFD Device, WFD Port, and WFD Pipeline

To select a WFD device, WFD port, and WFD pipeline,

1. Enumerate the IDs of the WFD devices with `wfdEnumerateDevices()` and select one of them.
2. Create a WFD device with the ID selected from step 1 by calling `wfdCreateDevice()`.
3. Enumerate the IDs of the WFD ports with `wfdEnumeratePorts()` and select one of them.
4. Create a WFD port with the ID selected from step 3 by calling `wfdCreatePort()`.
5. Query a list of `wfdPortMode` supported by the WFD port by calling `wfdGetPortModes()`.
6. Choose the desired `wfdPortMode`.

You can query the properties of the desired `wfdPortMode`, such as `WFD_PORT_MODE_WIDTH` and `WFD_PORT_MODE_HEIGHT`, by using `wfdGetPortModeAttrib*`().

7. Set the chosen `wfdPortMode` to the WFD port.
8. Enumerate the IDs of the WFD pipelines with `wfdEnumeratePipelines()` and select one of them.
9. Create a WFD pipeline with the ID selected from step 8 by calling `wfdCreatePipeline()`.
10. Commit the WFD pipeline with the WFD port by calling `wfdBindPipelineToPort()`.
11. Commit the state of the WFD port with `wfdDeviceCommit()` and the `WFD_COMMIT_ENTIRE_PORT` commit type.



Note: To exit SC7 mode (suspend to RAM), perform a modeset using `wfdDeviceCommit()` on the same `WFDPortMode` used before entering SC7 mode.

More Information

- > For details about APIs, see the [OpenWFD Specification](#).
- > For a list of addenda specific to NVIDIA, see the OpenWFD section of [NvDisplay](#).
- > For an example of using OpenWFD with an EGL device to interact with display, see the sample `openwfd_egldevice`.

6.3.2.10 Board-to-Display Connectors

For information about the connectors on your platform and the relationship between connectors and displays, see the documentation for your hardware.

6.3.3 X11 Window System

X is an application that is used to manage input devices, like a mouse and a keyboard, and the output devices like displays connected to a system. Any user application can communicate with the display graphics interfaces using different service routines.

Official documentation is at:

<https://www.x.org/wiki/Documentation/>

In the context of running graphic applications on top of X, in previous releases you could either have X running by default on the Ubuntu root filesystem started by the lightdm or gdm3 service, or start it manually. In this release, lightdm or gdm3 is not enabled by default. You must start X server manually.

6.3.3.1 Manually Starting X Server

6.3.3.1.1 To start the X server



Note:

The display configuration may be defined in `xorg.conf` before starting the server. After the server starts, use `xrandr` to set runtime configurations. For more information, see the [Using xrandr for Runtime Configuration](#).

Execute the following command:

```
sudo -b X -ac -noreset -nolisten tcp
```

The command line arguments may vary depending on platform.

6.3.3.1.2 To kill the X server

To get the pid of the X server, use:

```
ps aux | grep "X"
```

Then execute:

```
sudo kill <X server pid>
```

6.3.3.2 Runtime Configuration

Xorg provides several mechanisms that provide configuration and run-time parameters:

- > Command line options

- > Environment variables
- > `xorg.conf` and `xorg.conf.d` configuration files
- > Auto-detection
- > Fallback defaults

The SDK provides utilities for changing the configuration and run-time parameters:

- > `xrandr` utility—changes runtime parameters
- > `nvidia-xconfig` helper utility (tool)—assist in configuring `xorg.conf`



Note:

Customers who support hot plugging of an HDMI display must implement an X11 client that reconfigures the HDMI display upon receipt of an `RRScreenChangeNotify` X11 event. Typically, a `gnome-settings-daemon` is used for that purpose.

This solution is required to work around the X11 response to a hot-plugged HDMI display. During that response, X11 registers the HDMI output as connected but does not perform an automatic mode-set.

6.3.3.3 Using `xrandr` for Runtime Configuration

The NVIDIA Orin Driver supports the XRandR 1.2 (X11 Rotate and Resize) extension. This extension allows dynamic enabling, resizing, positioning, and orienting of displays. With the `xrandr` command line utility, you can control XRandR. That utility is included in the `x11-xserver-utils` package.

6.3.3.4 Querying Supported Monitors and Screen Resolutions

You can use `xrandr` to get information about supported monitors and screen resolutions.

6.3.3.4.1 To query attached displays and detect available modes

- > With the X server running, enter the following command in a terminal window:

```
xrandr
```

Output is similar to the following:

```
Screen 0: minimum 8 x 8, current 2560 x 1600, maximum 16384 x 16384
DP-0 connected primary 2560x1600+0+0 (normal left inverted right x axis y axis) 220mm x
140mm
    2560x1600      60.0*+
HDMI-0 disconnected (normal left inverted right x axis y axis)
```

6.3.3.5 Obtaining Additional Help

The `xrandr` utility provides a help menu that lists all supported options and provides guidance on their use.

6.3.3.5.1 To get further help and view all available options

- > In a terminal window, enter:

```
xrandr --help
```

6.3.3.6 Modifying the Static Configuration (Optional)

The minimal `xorg.conf` is installed on the target in the following location:

```
/etc/X11/xorg.conf
```

Additionally, directories of `*.conf` fragment files are also located in the following locations:

```
/etc/X11/xorg.conf.d/  
/usr/share/X11/xorg.conf.d/
```

By default, `xorg.conf` contains no specific settings for the screen resolution, virtual desktop size, bit depth, and display select, but instead query's the driver for the optimum settings based on displays enabled. It is the goal of the Tegra driver for the default settings along with runtime `xrandr` commands and X command-line arguments to be sufficient for most needs. In cases where defaults are not sufficient, appropriate settings for screen resolution, bit depth, and monitor enablement can be configured to create custom defaults. `xrandr` may still be used for runtime manipulation.



Note:

It is recommended that you use the `nvidia-xconfig` configuration tool to edit the `xorg.conf` file, rather than editing by hand. However, there may be circumstances where hand-editing is required.

6.3.3.7 Using nvidia-xconfig to Configure `xorg.conf`

The NVIDIA X configuration tool, `nvidia-xconfig`, simplifies the task of modifying your `xorg.conf` file. That tool parses the `xorg.conf` file, saves a backup, and then performs modifications to the configuration based on your choice of command-line options.

6.3.3.7.1 Getting Help on `nvidia-xconfig`

The `nvidia-xconfig` tool provides basic and advanced help.

6.3.3.7.1.1 To get basic help for `nvidia-xconfig`

- > In a terminal window, enter:

```
nvidia-xconfig --help
```

6.3.3.7.1.2 To get options for modifying `xorg.conf`

- > In a terminal window, enter:

```
nvidia-xconfig --advanced-help
```

6.3.3.7.2 nvidia-xconfig Usage Examples

The nvidia-xconfig tool provides many options. This section provides instructions on a small subset of the supported options.

6.3.3.7.3 Specifying a Custom EDID for the Monitor


Note:

If X11 does not recognize the mode list of a particular model of monitor, you may find that the monitor has an invalid extended display identification data (EDID). When that happens, the X driver cannot accurately determine the capabilities of the monitor.

6.3.3.7.3.1 To specify a custom EDID for the monitor

- > In a terminal window, enter:

```
nvidia-xconfig --custom-edid=HDMI-0:<path>
```

Where <path> is the complete path to edid.bin.

6.3.3.7.4 Setting Color Bit-Depth

You can use the nvidia-xconfig tool to set color bit-depth in the xorg.conf file.

6.3.3.7.4.1 To start X11 with a particular color bit-depth

- > In a terminal window, enter:

```
nvidia-xconfig --depth=<depth>
```

Where <depth> is one of the following supported color bit-depth values: 8, 15, 16, 24, and 30. For example:

```
nvidia-xconfig --depth=16
```

6.3.3.7.5 Specifying Modes

You can use the nvidia-xconfig tool to set display mode (resolution) in the xorg.conf file.

6.3.3.7.5.1 To add a mode to the mode list

- > In a terminal window, enter:

```
nvidia-xconfig --mode=<mode>
```

Where <mode> is the display mode expressed as X and Y pixels. For example:

```
nvidia-xconfig --mode="1024x768"
```

6.3.3.7.6 Enabling Debug Mode

You can use the nvidia-xconfig tool to specify a debug mode in the xorg.conf file. When debug mode is enabled, the X driver logs verbose details about mode validation. The driver logs these details in the X log file. You can use that information to troubleshoot mode database issues.

6.3.3.7.6.1 To enable debugging

- > In a terminal window, enter:

```
nvidia-xconfig --mode-debug
```

6.3.3.7.6.2 To disable debugging

- > In a terminal window, enter:

```
nvidia-xconfig --no-mode-debug
```

6.3.3.7.7 Multi-Display X Server Layout

For single-display systems, the default server layout is sufficient. However, for multi-display systems you must specify how the monitors are related to one another in the virtual space of the X11 desktop. Multi-Display is currently only supported with a single X screen at this time.

6.3.3.7.7.1 Enabling the One X Screen Layout

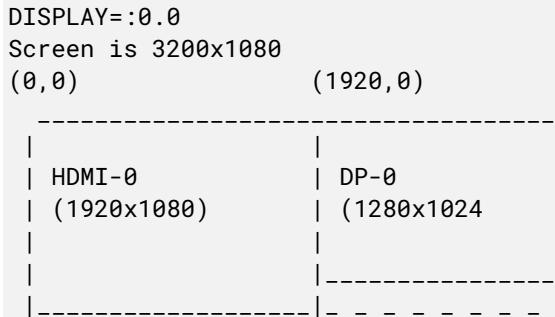
In the one X Screen configuration, multiple monitors will work together by displaying portions of the same X Screen, which is like a virtual desktop space. This is the default mode. Both mirroring and spanning layouts can be specified using one X Screen. The size of the X Screen is determined by the smallest rectangle that covers the extents of all the enabled displays in the chosen layout.

6.3.3.7.7.2 To enable one X Screen (default for span)

- > In a terminal window, enter:

```
nvidia-xconfig --only-one-x-screen
```

Here is a graphical depiction of a one X Screen side-by-side layout:



The xrandr command that produces the above settings is:

```
xrandr --output HDMI-0 --mode 1920x1080 --output DP-0 --mode 1280x1024 --right-of HDMI-0
```

6.3.3.7.7.3 To enable screen mirroring

- > In a terminal window, enter:

```
nvidia-xconfig --metamode=orientation=clone
```

6.3.3.7.7.4 To enable screen spanning

- > In a terminal window, enter:

```
nvidia-xconfig --metamode=orientation=<value>
```

Where <value> can be:

- > RightOf (the default)
- > LeftOf
- > Above
- > Below

6.3.3.8 Modifying xorg.conf

This section describes xorg.conf changes that require you to make manual modifications, such as for enabling screen saver features, EDID polling, blending, and video overlays.

6.3.3.8.1 Enabling Screen Saver Features

By default, X11 features for screen blanking, suspending, and disabling display are all disabled. However, you can enable those features.

6.3.3.8.1.1 To re-enable any of these screen saver related features

- > Modify or remove the following file:

```
/etc/X11/xorg.conf.d/disable_screensaver.conf
```

6.3.3.8.2 Enabling EDID Polling and Native Resolution

Instead of statically setting the resolution used in the configuration file, the EDID modes can be polled by the driver and the native resolution of the display will be used. EDID polling is always enabled in the Tegra driver. The information in the [Specifying Modes](#) subtopic can still be used to manually add specific modes. Remove these modes to only use EDID modes.

6.3.3.8.3 Enabling Blending and Video Overlays

TegraOverlayPriority is a 32-bit integer that is used to control overlay stacking order. The overlay with the lowest depth is in front of all others. This value has meaning only when multiple overlays are present on a display. This value can range between 0 and 255 (both values inclusive), where the default is 255.

TegraOverlayBlendmode determines how the X overlay is combined with the overlay behind it during scanout. Available modes are:

- > Opaque (default)
- > SourceAlphaBlend
- > PremultSourceAlphaBlend

This value has meaning only when an external process has created a display that is behind the X server.

6.3.3.8.3.1 To set these overlay attributes

- > In the "Device" section, add the following line (with desired values):

```
Option "MetaModes" "nvidia-auto-select {
```

```
TegraOverlayBlendmode = PremultSourceAlphaBlend,  
TegraOverlayPriority = 0 }"
```

Here nvidia-auto-select is the default mode name and can be replaced with the desired mode name.

These properties are per-output and can also be queried/modified during run-time via xrandr.

6.3.3.8.3.2 To query the current values

- Execute the following command:

```
xrandr --prop
```

6.3.3.8.3.3 To modify these properties

- Execute the following commands:

```
xrandr --output HDMI-0 --set TegraOverlayPriority 0  
xrandr --output HDMI-0 --set TegraOverlayBlendmode PremultSourceAlphaBlend
```

**Note:**

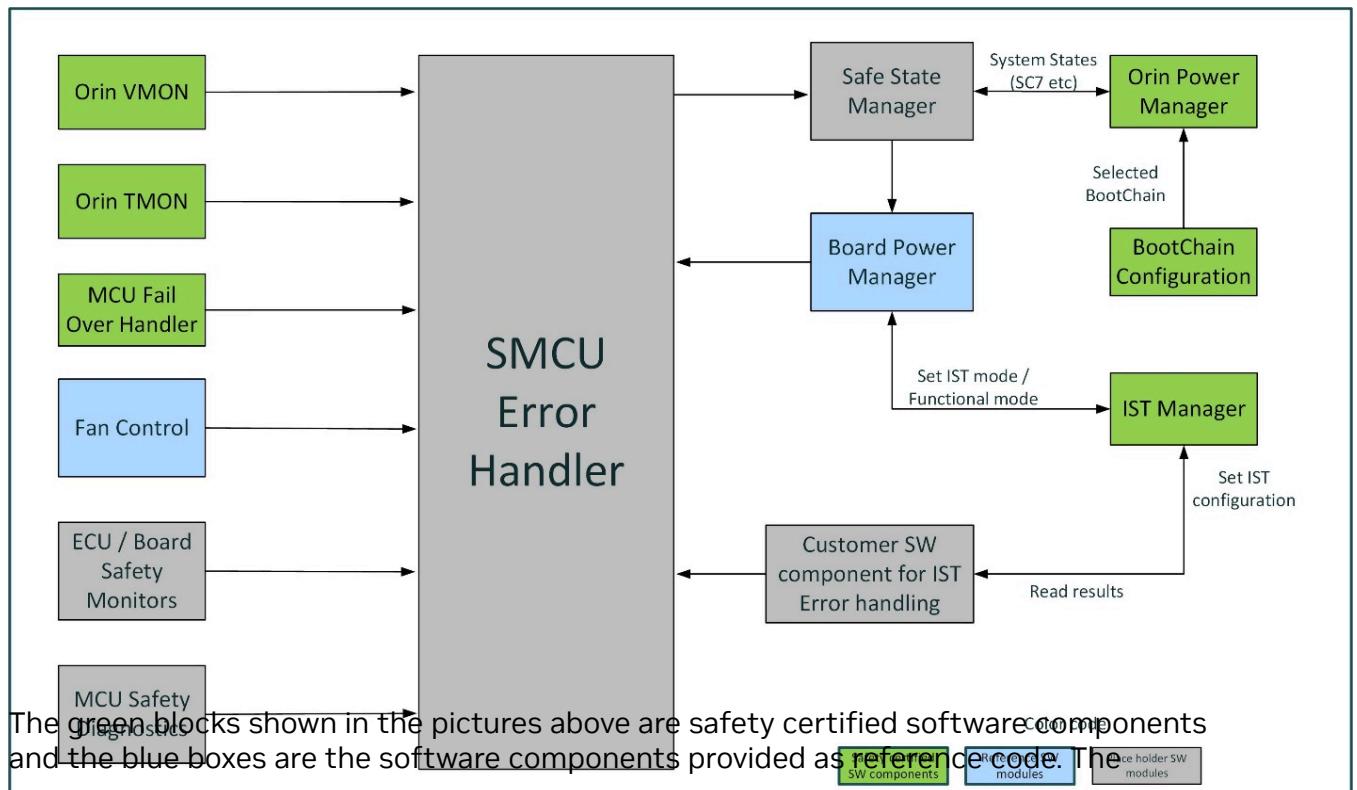
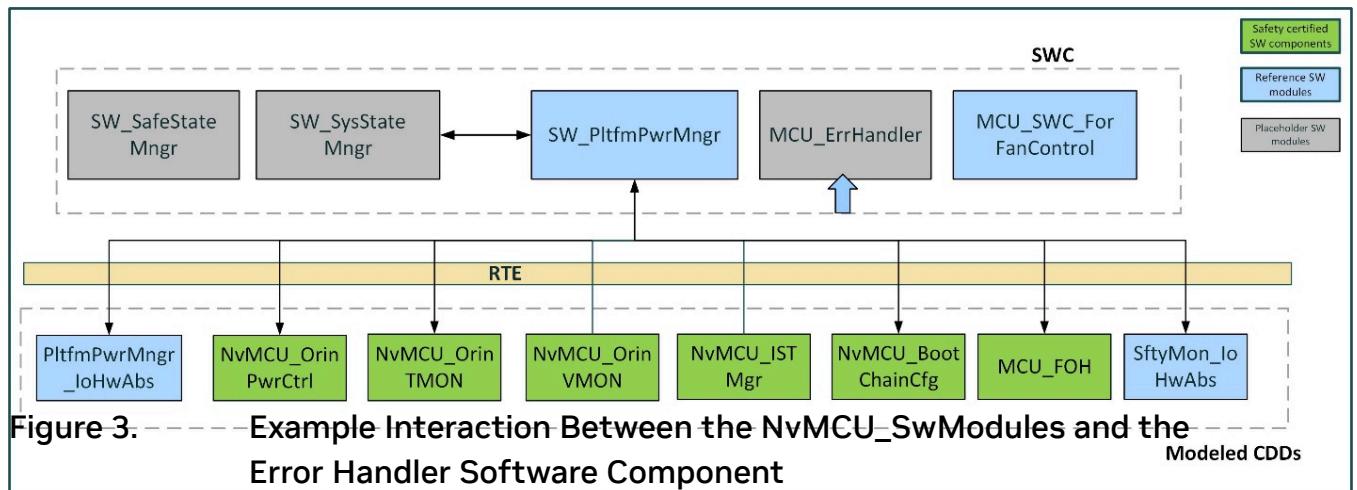
X does not have a notion of alpha blending, so all alpha blending on the enabled display must go through OpenGL ES rendering. Alpha blending for natively rendered pixels is undefined.

6.4 MCU Software Modules

6.4.1 High-Level Architecture

Here is a high-level Architecture diagram, which shows the different functionalities provided by NvMCU_SwModules from NVIDIA DRIVE® OS release, and the static architecture of customer developed software components.

Figure 2. NVIDIA DRIVE OS MCU Software High-Level Architecture



gray boxes indicate placeholder software components used to evaluate the software architecture of NvMCU_SwModules. You must develop the complete functionality of reference software components and the placeholder software modules. The following topics describe the functionality provided by the NvMCU_SwModules and Reference software components.

6.4.2 Platform (Board) Power State Management

- Platform Power Manager decides the complete power-on/off sequence for the board, and implements the steps recommended by the HW design.
- Invokes the interfaces and services provided by other software modules of NvMCU_SwModules in the required order as per the system requirements. As a part of the Power sequence, it triggers IST, programs the VMON and TMON thresholds, and executes safety related checks using the interfaces provided by NvMCU_SwModules.
- Provides interfaces to power on and power off the board from customer applications. The interfaces are used by shell commands, which execute the power-on/off based on the user inputs on MCU serial console.
- Provides interfaces for mode change requests, such as Orin-Reset, MCU-reset, force-shutdown, and so on.
- Detects failures during board power sequence and reports to Error Handler module.
- NVIDIA reference code will NOT include the following features:
 - MCU shut-down and wake-up using TLF
- SW_PlatformPwrMgr (Application), which implements a high-level state-machine for the power management function. SW_PlatformPwrMgr facilitates power-on, power-off, and force shutdown of NVIDIA DRIVE Orin™ SoCs based on events triggered by Hardware (wake-up, KL15 ...) and software (IST, Safe shutdown, User commands...) inputs. It also supports the platform reset through MCU reset.
- PlatformPwrMgr_IoHwAbs (CDD), which provides the services to SW_PlatformPwrMgr primarily related to platform power control, Ethernet initialization, network communication, and so on.

Safe shutdown: The safe shutdown sequence is provided as a reference implementation. When requested for safe-shutdown of the board, this module waits for safe shutdown indicator signal over GPIO from NVIDIA Orin and continues to power-off the board. If the GPIO is not asserted, a configurable timeout is provided to avoid indefinite waiting.

SC7: NVIDIA Orin suspend mode is supported with a few shell commands from MCU. Entry and Exit SC7 commands are provided for verification of the functionality.

6.4.3 NVIDIA Orin Power Management

- Provides an interface to power on and power off the NVIDIA DRIVE Orin™ SoC.
- Provides an interface to set the GPIO pins for Boot-chain of NVIDIA Orin SoC while powering on.

- > This is a stateless (independent of IST, SC7) Component as it acts on the commands from other SWCs on MCU, which might hold the states and state management.
- > Detects HW Error scenarios like HW power-up failure detection via PG Signal from VRS10/11/12 and NVIDIA Orin rails, and the power-on is aborted.
- > Implements safety-related latent fault detection requirements for VRS10 and VRS11.
- > Supports safe shutdown of NVIDIA Orin.
 - During power-Off/ Reset request, Power Manager waits for Handshake GPIO to be set before powering off the VRS10.
 - A timeout of 20 seconds is provided in case the GPIO is not asserted.
- > Needs a Platform Power Management Software Component (provided as a reference code), which invokes the APIs provided by NVIDIA Orin Power Manager. The same requests are provided by the shell commands also.
- > Provides interfaces for SC7 entry and exit transitions. Implements the hardware-level triggers to manage the SC7 entry and exit sequence of NVIDIA Orin.

6.4.4 NVIDIA Orin Voltage Monitoring via VRS12 (VMON)

- > Initial configuration and set up for VMON, executing the HW BIST provided by VMON
- > Programming the VMON Chip with thresholds for UV/OV detection
- > Continuous monitoring of VMON Reset pin indicating UV/OV and reporting the detected errors to Error handler module
- > Implements safety measures like toggle check for VMON NIRQ pin, ACT/SHDN and ACT/SLP signals.
- > Monitors the power-up and power-down sequences for NVIDIA DRIVE Orin™ SoC rails.
- > The following error scenarios are detected
 - VMON Chip internal errors are detected via BIST failure
 - I2C communication failure while read/write operations are detected via CRC
 - Errors during the programming of VMON chip are detected by reading back the values written to the VMON registers.
- > In the current SW functionality, VMON error are notified to Error handler module, and the board is powered off.

6.4.5 NVIDIA Orin Temperature Monitoring

NVIDIA MCU software modules in NVIDIA DRIVE Orin™ supports NVIDIA Orin temperature monitoring by the SoC TMON temperature sensor (TMP451).



Note: Board temperature monitor (External ECU TMON) via TMP451 sensor on the board is not provided by NvMCU_SwModules. On NVIDIA DRIVE Orin™ boards, the scope is limited to Orin Temperature monitoring only.

- > Programs the ALERT and SHUTDOWN thresholds for SoC TMON.

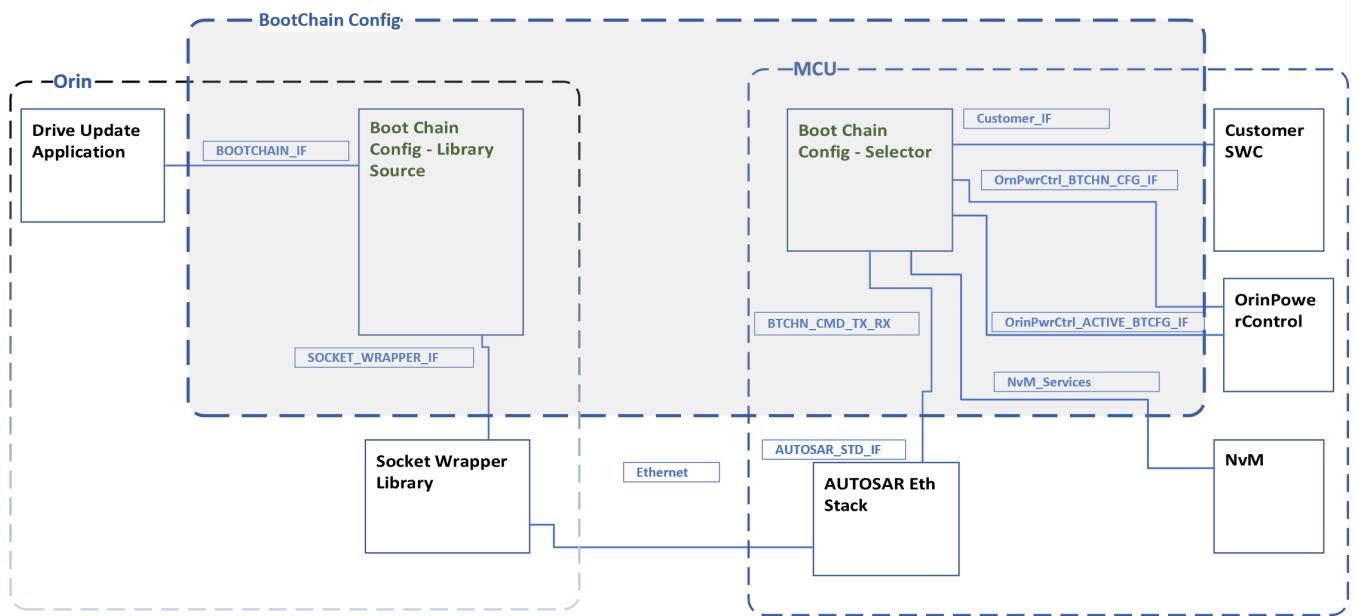
- > Monitors the THERM_ALERT and THERM_SHDN signals from NVIDIA Orin SoC temperature sensor over GPIO lines.
- > Any thermal events detected will send a notification to the Error Handler module.
- > Detects the following hardware error scenarios:
 - Errors in TMON configuration via I2C are detected by read-back after write mechanism.
 - Thermal failures of NVIDIA Orin SoC are detected via the THERM_SHDN and THERM_ALERT signals.
 - Stuck-at fault problems are detected by performing toggle check for THERM_ALERT and THERM_SHDN GPIO lines.
- > MCU software should not read the temperature or status registers from TMP451 over I2C, after the Orin boots-up. This is due to the restrictions from multimaster I2C.

6.4.6 NVIDIA Orin Boot Chain Configuration Support

- > In NVIDIA DRIVE AGX Orin™, NVIDIA DRIVE® Update application running on Orin manages software updates of NVIDIA Orin reliably by maintaining multiple Boot Chains (Boot Chain A, B, C). Boot Chain Config module facilitates GPIO based Boot Chain selection and reboots Orin in selected Boot Chain.
- > Boot Chain Config module is decomposed into two sub modules-Boot Chain Config Library Source and Boot Chain Config Selector and they run on Orin and MCU, respectively.

- > The following diagram gives overview of interfaces and positioning of Boot Chain Config modules in the system.

Figure 4. Overview of Bootchain Configuration Functionality



On NVIDIA Orin, submodule Boot Chain Config Library Source offers API to facilitate the following features:

- > Select default Boot Chain configuration.
- > Select next Boot Chain.
- > Get Default Boot Chain configuration.
- > Get Active boot Chain configuration.
- > Perform reboot of Orin and MCU to boot in selected Boot Chain.

Each of the preceding features needs support of BootChain Config Selector running on MCU to complete operations behind.

Implementation on NVIDIA Orin Side

NVIDIA DRIVE Update/user applications intending to use these features shall link to Boot Chain Config Library (`libmcu_common_if.so`).

Boot Chain Config Library (`libmcu_common_if.so`) formulates the Request-command, forwards to MCU, and wait for the Response-command. On receipt of the Response-command from MCU, the Boot Chain library processes the response and returns to caller.

The library (`libmcu_common_if.so`) uses Socket Wrapper library for transmission and reception of commands over UDP protocol.

The following parameters are configurable and are maintained in tacp configuration file. Appropriate changes must be made at MCU if the parameter values are altered

IP address of MCU: AURIX_IP_ADDRESS=10.42.0.146

IP address of Tegra A: TEGRA_A_IP_ADDRESS=10.42.0.28

Server Port on MCU: AURIX_BOOTCHAIN_PORT=5001

Implementation at MCU Side

On MCU, UDP packet is evaluated to check if the received packet is Boot Chain Request-command. API from Boot Chain Config Selector is invoked to further process the packet and perform appropriate operation.

Boot Chain Config Selector validates Request-command, performs appropriate operation on valid request, and triggers Response-command transmission.

Boot Chain Config Selector depended on the following modules to perform requested operation.

- > NvM (persistent memory) : To store Boot Chain configuration data
- > Customer SWC: To handle reboot/reset request
- > NVIDIA Orin power control: To get active boot chain (Chain A/B/C)

These are the Socket properties for Boot Chain Config.

MCU IP Address: 10.42.0.146

Orin-A IP Address: 10.42.0.28

VLAN Id: 200

Port: 5001

6.4.7 NVIDIA Orin IST Manager

NVIDIA DRIVE Orin™ IST Manager

- > Provides an interface to set the IST Configuration. Communicates the configuration set by the user application to NVIDIA DRIVE Orin™ software (IST Client) over Ethernet (for Key-On/OFF Orin IST).
- > Provides an interface to read the IST results.
- > Provides interfaces using which the Platform Power manager can query if a Keyon/KeyOff IST must be executed, and query whether IST is completed.
- > During bootup, decides whether NVIDIA Orin should be in IST Mode or Normal mode.
- > Triggers IST by GPIO assertion, monitors the execution of IST on NVIDIA Orin and detects IST stuck scenarios through timeouts.
- > Provides an interface to abort the IST execution.
- > Detects and handles the following HW Error scenarios.
 - IST stuck is detected by a GPT-based HW timer with a configurable timeout.
 - Communication with NVIDIA DRIVE Orin™ SoC is monitored by a software timer with a configurable timeout.

- > Depends on an active Ethernet (VLAN200) connection between MCU and NVIDIA Orin.
- > The customer application is expected to set a valid IST configuration during every power-cycle, using the interfaces provided by NvMCU_ISTManger. This is done using the Serial console Shell Commands.

6.4.8 Failover Handler on MCU

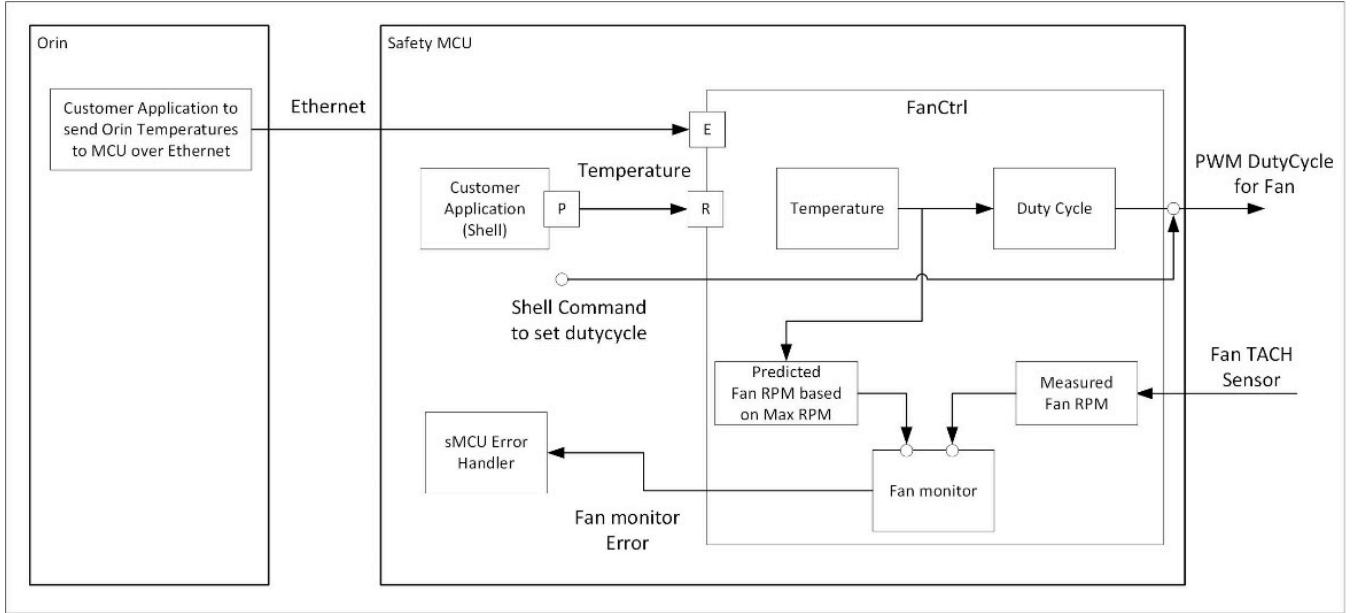
- > Monitors the SOC_ERROR pin (operated in Toggle mode) used to assert NVIDIA DRIVE Orin™ Safety Status and reports any detected error to Error Handler module.
- > Communicates with FSI on NVIDIA Orin over SPI 2 as initiator to get Orin Failure information
- > Notifies the MCU error handler module to initiate Fail-Over, when a critical error is reported by NVIDIA Orin FSI.
- > Passes on the error status received in the periodic SPI frames to Customer application for persistent storage.
- > Provides an interface to notify periodic failure status received from System Error Handler of Orin-FSI, along with Key of Seed from FSI
- > Provides an interface to notify Orin SOC Error Pin Status
- > Provides an interface to send periodic Seed to FSI
- > Handles the following HW Error scenarios
 - SOC_ERROR stuck at in Toggle mode
 - SPI Channel errors

6.4.9 Fan Control and Monitoring

Fan-control functionality is provided as a reference software to demonstrate the control of Fan speed based on the temperature of NVIDIA DRIVE Orin™ SoC. Measured Orin temperature values are sent to MCU over Ethernet. The Fan Control reference SW component on MCU computes the PWM duty cycle required to drive the Fan, based on the temperature data using a characteristic curve. In addition, the Fan-Tach sensor signal

from the Fan is read by MCU software periodically and fan speed is monitored against the requested RPM to detect faults in Fan.

Figure 5. Fan Control Overview



The following functionalities are provided by the Fan-Control reference SW component -

- > Receives the Ethernet frames with Orin temperature data and selects the temperature for controlling Fan speed. The reference design chooses the maximum of the three values received over Ethernet. The Customer can extend this feature in the function `MCU_SWC_ForFanControl_ChOOSETemperature()`.
- > When the data is not available over Ethernet, or if the data is corrupted, another application on MCU can set the temperature over an AUTOSAR interface. If the temperature is provided by the SW component on MCU, this value will be used to control the Fan PWM for the rest of the power-cycle.
- > The temperature value is converted into the desired PWM duty cycle using the characteristic curve and sent to the HW.



Note: When the FanCtl software functionality is disabled or not yet integrated, there is a shell command `set_fanpwm` which directly sets the PWM duty-cycle value on the Fan PWM hardware. The shell command `show_fanrpm` is provided to readout the Fan speed and the duty-cycle from MCU serial console.

- > During initialization, the Fan is operated at full speed, by applying 100% PWM signal (`MCU_SWC_FANCTRL_MAX_PWM`) and maximum possible Fan speed is measured. It is compared with the rated speed of Fan as per the Manufacturer's specifications and large deviations are considered as permanent degradation of Fan.
- > During normal operation, the actual speed of Fan measured from the Tach sensor is compared with the desired speed. Large deviations are considered as failures and reported on the MCU console. Because the desired speed of operation depends on many external parameters such as ambient temperature, altitude, and so on, it

cannot be computed precisely. For monitoring, an approximated value calculated based on the maximum speed of operation measured during that power-on cycle, is used.

Receiving Temperature Values from NVIDIA Orin

A sample application on Orin CCPLEX sends the Temperature values periodically (1 second) to MCU over Ethernet VLAN200. This application is packaged in On Standard build only safety build will operate Fan from MCU side software

The temperature data received over Ethernet will have the following structure.

ID	T1 : Orin Internal Temperature	T2 : Orin TMP451 reading	T3 : Board TMP451 reading	T4 : Reserved	Checksum
Size = 1 byte Value : 0x01 Indicates Fan Control data	Size = 4 bytes	Size = 4 bytes	Size = 4 bytes	Size = 4 bytes Not used in the current SW	Size = 1 byte

- > The temperature values have a resolution of millidegree Celsius. For example, a 78.527 degree Celsius is sent as 78,527 decimals, which is 0x000132BF (hex). The receiving module shall convert the data into actual temperature by dividing the data by 1000U.
- > After power-on, the FanControl software waits for Orin temperature data over Ethernet for a configurable time duration. If Ethernet packets are not received, a safe PWM duty cycle is applied to prevent the board from overheating.
- > The communication over Ethernet is protected by a checksum. During runtime, if Ethernet frames are not received or received with invalid data, the fan control switches to a safe PWM duty cycle.

6.4.10 Common Interface

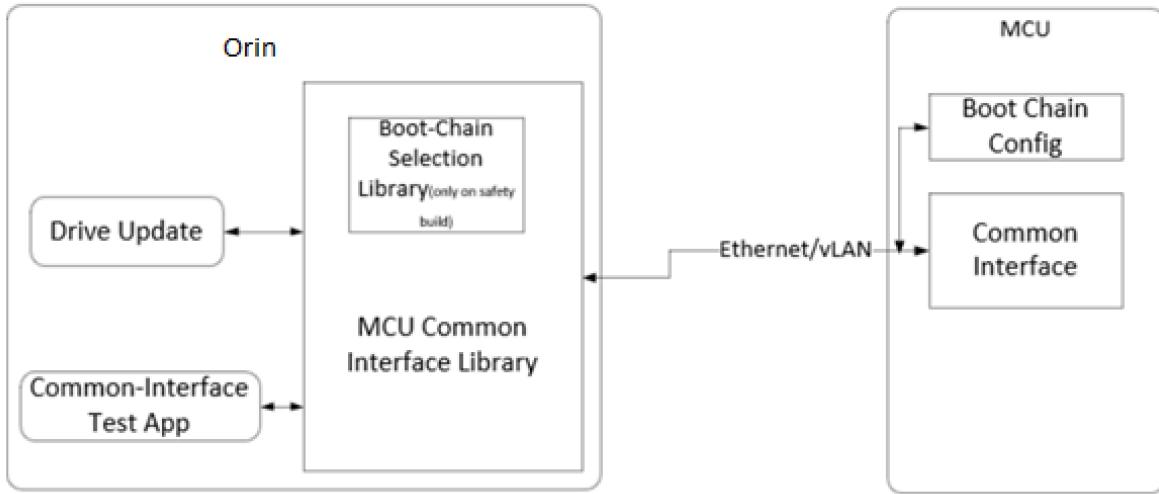
In NVIDIA DRIVE AGX Orin™ platform, NVIDIA Orin communicates with MCU using UDP protocol over Ethernet for update of MCU firmware and its configuration.

Configuration of MCU and firmware update includes:

- > Setting/Reading network configuration
- > Reading current version information of MCU FW
- > Reading InfoROM content (If supported by HW)
- > Compatibility validation of MCU FW to be flashed
- > Flash programming update and production MCU FW
- > Triggering SC7 entry

All these functionalities are realized under module Common Interface.

The following diagram provides positioning of Common Interface modules in the system:



Implementation on NVIDIA Orin Side

NVIDIA Orin initiates request for configuration update using APIs offered by Common Interface.

User applications intending to use these features shall link to Common Interface Library (libmcu_common_if.so).

Common Interface Library (libmcu_common_if.so) formulates Request-command, forwards to MCU, and waits for Response-command. On receiving the Response-command from MCU, the common interface library processes the response and return to caller.

The library (libmcu_common_if.so) uses Socket Wrapper library for transmission and reception of commands over UDP protocol.

The following parameters are configurable and are maintained in tacp configuration file. Appropriate changes must be made at MCU if the parameter values are altered.

IP address of MCU: AURIX_IP_ADDRESS=10.42.0.146

IP address of Tegra A: TEGRA_A_IP_ADDRESS=10.42.0.28

Server Port on MCU: AURIX_BOOTCHAIN_PORT=5001

VLAN ID : 200



Note: As library (libmcu_common_if.so) is common for both BootChain APIs and CommonIf APIs, the preceding configuration is common for both.

Implementation at MCU Side

Common Interface on MCU is realized as an unmodeled CDD and it leverages interface of SWC Boot Chain Config for transmission of message. API from Common Interface is called from Boot Chain Config if received data belongs to common interface. Common Interface further processes the received data and perform appropriate operation.

These are the Socket properties for Common Interface:

MCU IP Address : 10.42.0.146

Orin-A IP Address : 10.42.0.28

Vlan Id: 200

Port: 5001

6.4.11 MCU Communication Coordinator Daemon

This is the MCU communication coordinator (MCC) daemon sample application.



Note: It is a sample implementation and uses Socket APIs for communication with MCU. Because the MCC Daemon and libraries are sample implementation, users of NVIDIA DRIVE OS is expected to implement safety and security provisions for it based on the chosen medium of communication.

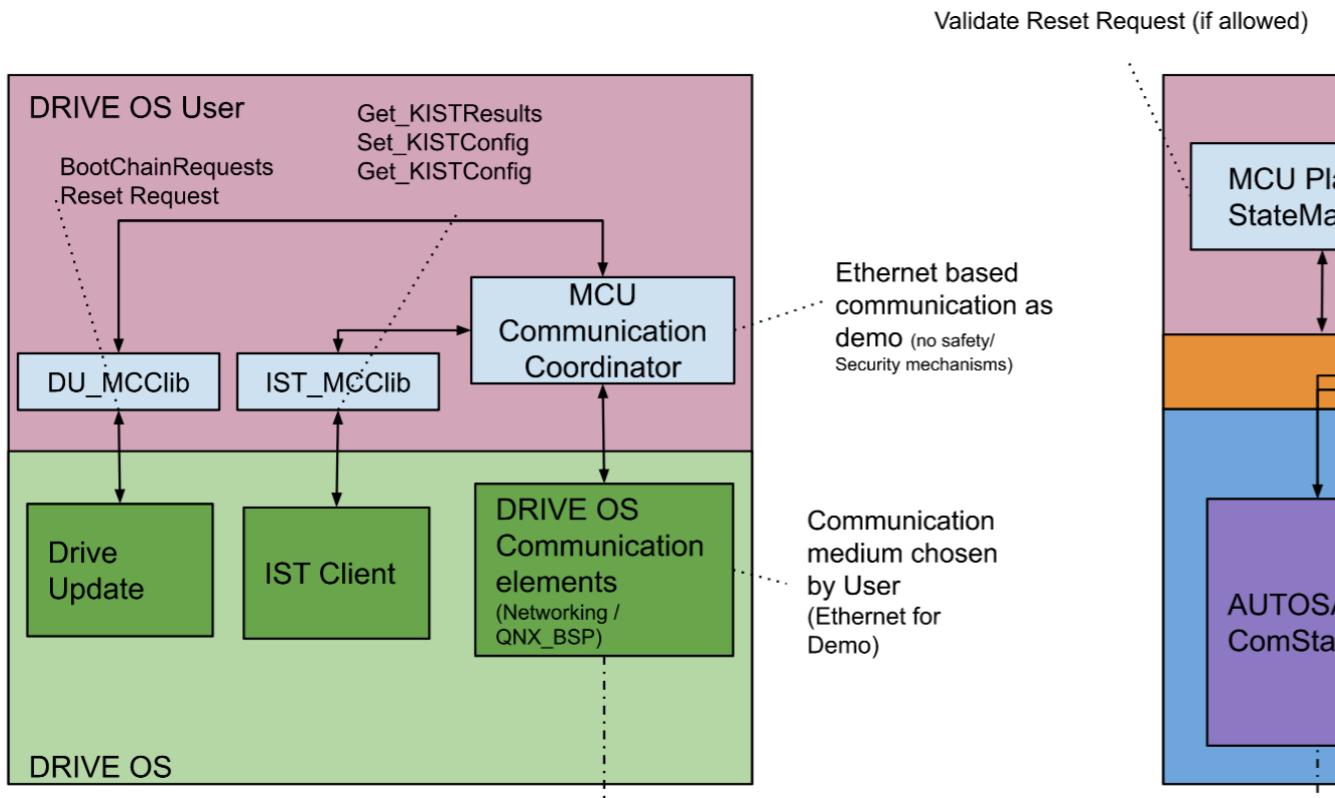
Functionalities

These are the functionalities of the MCU communication coordinator daemon process (MCC daemon):

- > Relays the messages and responses among IST Client, DRIVE Update, and Common IF clients on NVIDIA DRIVE Orin™ SoC and MCU.
- > Communicates with clients on NVIDIA DRIVE Orin™ SoC using NvScilpc.

- > Communicates with the MCU side using UDP socket communication.

Figure 6. Overview of NVIDIA Orin MCU Communication



Design Aspects

Implementation on NVIDIA DRIVE Orin™ SoC

IST Client, DRIVE Update, and Common IF clients can send a message to the MCC daemon over their respective NvScilpc channels, and the daemon relays the message over the corresponding UDP socket. For DRIVE Update and Common IF client messages, it waits for a response from the UDP socket. Upon receipt of the response, the daemon writes it to the corresponding NvScilpc channel.

Another thread of the MCC daemon is dedicated to waiting for messages on UDP socket for any communication from IST Manager on MCU. Once received, it writes the message to NvScilpc channel corresponding to IST Client.

In case of timeout or errors, fixed number of retries are done.

The following are the configuration parameters:

- > UDP socket parameters taken from the tacp.cfg file:
 - AURIX_IP_ADDRESS=10.42.0.146
 - AURIX_SERVER_PORT=5000

- AURIX_BOOTCHAIN_PORT=5001
- CLIENT_IP_ADDRESS=10.42.0.28
- > Default timeout: 1 second
- > Default number of retries during init time: 120
- > Default number of retries during the runtime: 5
 - Note that clients should wait for more than 5 * 1 seconds to assume request might have failed. For example, 7 retries with 1 second timeout.
- > NvScilpc channels:
 - For IST: nvmcc_ist_ipc_0, nvmcc_ist_ipc_1
 - For DU: nvmcc_du_ipc_0, nvmcc_du_ipc_1
 - For Common IF: nvmcc_cif_ipc_0, nvmcc_cif_ipc_1
 - As a convention: The daemon should open <nvsciiipc_ep_name>_0 endpoint and clients should open corresponding <nvsciiipc_ep_name>_1 endpoint.

Linux-Specific Settings

Ensure that NvScilpc and socket (tacp) services are marked as dependency before launching the MCC daemon.

Source Path

```
<PDK>/samples/mcc_daemon
```

6.4.12 MCU Communication Coordinator for IST Client

The IST client MCU communication coordinator (ist_client_mcc) provides APIs enabling the IST client to communicate with IST manager.

Functionalities

- > Provide an interface for communication between the IST client running on NVIDIA DRIVE Orin™ SoC and IST manager running on MCU.
- > Communication frame format and protocol to pass messages between MCU and IST client are defined in this module.

APIs

The following APIs must be implemented and exported in libist_client_mcc.so:

```
/**  
 * Do everything needed for MCC and send KIST result  
 *  
 * @param [in] result: KIST result data  
 * @returns 0 on success  
 */  
extern int ISTClient_mcc_send_results(ist_client_result_t result);
```

Types

Types defined in `ist_client_mcc.h`:

```
/**  
 * IST diagnostic result structure  
 */  
typedef struct ist_client_result {  
    uint8_t hw_result;      /**< hw_result value (opaque to mcc) */  
    uint8_t reserved_0;     /**< reserved_0 value (opaque to mcc) */  
    uint8_t sw_rpl_status;  /**< sw_rpl_status value (opaque to mcc) */  
    uint8_t sw_preist_status; /**< sw_preist_status value (opaque to mcc) */  
} ist_client_result_t;
```

Usage

The following pseudo code is provided to illustrate how the `ist_client` uses the APIs provided by the `ist_client_mcc` library.

```
static ist_client_result_t g_ist_client_keyist_result = {0};  
  
int ist_client_main(int argc, char *argv[])  
{  
    /* Store result in g_ist_client_keyist_result */  
  
    /* Send results to MCU */  
    ISTClient_mcc_send_results(g_ist_client_keyist_result);  
  
    return 0;  
}
```

6.5 NvDisplay

NvDisplay is the display architecture supported on Linux starting in NVIDIA DRIVE® OS 6.0.

SPI provides a Debian package file for you to install and build, which contains SPI source code. If you need a package of SPI source code, contact your NVIDIA representative and the customer service team will provide detailed support.

6.5.1 Components

The following are NvDisplay components and their libraries:

- > OpenWFD (`libtegrawfd.so`)
- > NvKms (`nvidia-modeset.ko`)
- > Serializer (`maxim_gmsl_dp_serializer.ko/ti_fpdlink_dp_serializer.ko`)

6.5.2 OpenWFD

OpenWFD (Open Windowing Foundation Display) is a Khronos API that provides a low-level hardware abstraction interface for windowing systems and applications to make use of display hardware.

Applications that need to interact with NvDisplay should do so via the OpenWFD API.

The specification of the OpenWFD API is available at Khronos' registry at https://www.khronos.org/registry/OpenWF/specs/OpenWF_Display_1_0_Specification.pdf (hereby referred to as the spec).

The NvDisplay architecture on NVIDIA DRIVE[®] OS includes an OpenWFD driver (`libtegrawfd.so` – hereby referred to as the WFD driver) that supports a subset of the core OpenWFD APIs listed in the spec along with additional NVIDIA specific extensions.

Usage guidelines for the core OpenWFD APIs can be obtained from the spec and any deviations from the spec or additional details will be described in this document.

6.5.2.1 Supported OpenWFD APIs

The following table lists all the core OpenWFD APIs and the available support for them on the WFD driver.

API	Support in <code>libtegrawfd.so</code>
<code>wfdGetStrings</code>	Supported on Linux build
<code>wfdIsExtensionSupported</code>	Supported on Linux build
<code>wfdGetError</code>	Supported on Linux build
<code>wfdEnumerateDevices</code>	Supported on Linux build
<code>wfdCreateDevice</code>	Supported on Linux build
<code>wfdDestroyDevice</code>	Supported on Linux build
<code>wfdDeviceCommit</code>	Supported on Linux build
<code>wfdGetDeviceAttribi</code>	Supported on Linux build
<code>wfdSetDeviceAttribi</code>	Supported on Linux build
<code>wfdCreateEvent</code>	Not supported
<code>wfdDestroyEvent</code>	Not supported
<code>wfdGetEventAttribi</code>	Not supported
<code>wfdDeviceEventAsync</code>	Not supported

API	Support in libtegrawfd.so
wfdDeviceEventWait	Not supported
wfdDeviceEventFilter	Not supported
wfdEnumeratePorts	Supported on Linux build
wfdCreatePort	Supported on Linux build
wfdDestroyPort	Supported on Linux build
wfdGetPortModes	Supported on Linux build
wfdGetPortModeAttrib{i/f}	Supported on Linux build
wfdSetPortMode	Supported on Linux build
wfdGetCurrentPortMode	Supported on Linux build
wfdGetPortAttribi	Supported on Linux build
wfdGetPortAttribf	Supported on Linux build
wfdGetPortAttribiv	Supported on Linux build
wfdGetPortAttribfv	Supported on Linux build
wfdSetPortAttribi	Supported on Linux build
wfdSetPortAttribf	Supported on Linux build
wfdSetPortAttribiv	Supported on Linux build
wfdSetPortAttribfv	Supported on Linux build
wfdBindPipelineToPort	Supported on Linux build
wfdGetDisplayDataFormats	Not supported
wfdGetDisplayData	Not supported
wfdEnumeratePipelines	Supported on Linux build
wfdCreatePipeline	Supported on Linux build
wfdDestroyPipeline	Supported on Linux build
wfdCreateSourceFromImage	Supported on Linux build
wfdCreateSourceFromStream	Not supported

API	Support in libtegrawfd.so
wfdDestroySource	Supported on Linux build
wfdCreateMaskFromImage	Not supported
wfdCreateMaskFromStream	Not supported
wfdDestroyMask	Not supported
wfdBindSourceToPipeline	Supported on Linux build
wfdBindMaskToPipeline	Not supported
wfdGetPipelineAttribi	Supported on Linux build
wfdGetPipelineAttribf	Supported on Linux build
wfdGetPipelineAttribiv	Supported on Linux build
wfdGetPipelineAttribfv	Supported on Linux build
wfdSetPipelineAttribi	Supported on Linux build
wfdSetPipelineAttribf	Supported on Linux build
wfdSetPipelineAttribiv	Supported on Linux build
wfdSetPipelineAttribfv	Supported on Linux build
wfdGetPipelineTransparency	Supported on Linux build
wfdSetPipelineTSCColor	Not supported
wfdGetPipelineLayerOrder	Supported on Linux build

The following OpenWFD attributes have limited support for querying:

- WFD_PORT_BACKGROUND_COLOR – Cannot be queried via wfdGetPortAttribi API. wfdGetPortAttribiv and wfdGetPortAttribfv APIs can still be used for querying WFD_PORT_BACKGROUND_COLOR.

6.5.2.2 Supported OpenWFD Extensions

The following OpenWFD extensions are supported by the OpenWFD Driver.

```
WFD_NVX_create_source_from_nvscibuf
WFD_NVX_commit_non_blocking
WFD_NVX_nvscisync
WFD_NVX_port_mode_timings
```

These extensions are described in the *NVIDIA DRIVE® OS 6.0 SDK API Reference*.

6.5.2.3 OpenWFD Usage Guidelines

- > The macro `WFD_NVX_create_source_from_nvscibuf` must be defined before including the headers `wfd.h` and `wfdext.h` for enabling the `WFD_NVX_create_source_from_nvscibuf` extension.
- > The macro `WFD_WFDEXT_PROTOTYPES` must be defined before including the `wfd.h` and `wfdext.h` headers.
- > To clear the contents on a `WFDPipeline` (performing a null flip), `wfdBindSourceToPipeline` must be called with `0` as the argument for `WFDSOURCE` before calling `wfdDeviceCommit` on the `WFDPipeline`. This is demonstrated in the `openwfd_nvsci_sample` application present in the SDK.
- > During the deinitialization of OpenWFD applications, you must follow these steps:
 1. Perform a null flip on all `WFDPorts` and `WFDPipelines` that were previously bound to a `WFDSOURCE`.
 - If nonblocking commits are being used and `WFD_PIPELINE_POSTFENCE_SCANOUT_BEGIN_NVX` is set to `WFD_TRUE`, the post-flip fence of the null flip commit should be waited on.
 - If nonblocking commits are being used and `WFD_PIPELINE_POSTFENCE_SCANOUT_BEGIN_NVX` is set to `WFD_FALSE`, the post-flip fence of the `wfdDeviceCommit` call before the null flip commit should be waited on.
 2. Destroy all `WFDSOURCE` objects by invoking `wfdDestroySource` on them only after all null flip commits are complete. Blocking `wfdDeviceCommit` calls ensures that the null flip is complete when the `wfdDeviceCommit` call returns. For nonblocking `wfdDeviceCommit` calls, synchronization must be handled by the OpenWFD client as described previously.
 3. Destroy all `WFDPipeline` and `WFDPort` objects by using `wfdDestroyPipeline` and `wfdDestroyPort`, respectively, before finally destroying the `WFDDevice` object using `wfdDestroyDevice`.
- > For committing to multiple `WFDPipeline` handles bound to the same `WFDPort`, it is recommended to call `wfdDeviceCommit*` with `WFDCOMMITTYPE` set to `WFD_COMMIT_ENTIRE_PORT`.
- > When you use `WFD_COMMIT_ENTIRE_PORT` in a `wfdDeviceCommit` call, all the `WFDPipeline` objects part of the commit must either have a surface bound(normal flip) or have no surfaces bound (null flip). A mix of normal and a null flip in one `WFD_COMMIT_ENTIRE_PORT` `wfdDeviceCommit` call is not allowed due to a limitation on the NvDisplay resource manager. To ensure that any errors are caught, the OpenWFD API `wfdGetError` must be invoked after calls to the following OpenWFD APIs that take in a `WFDDevice` handle as one of the inputs and return void:
 - `wfdDeviceCommit`
 - `wfdDeviceCommitWithNvSciSyncFenceNVX`
 - `wfdDestroyPort`
 - `wfdSetPortMode`
 - `wfdGetPortAttrib{iv/fv}`
 - `wfdSetPortAttrib{i/v/iv/fv}`
 - `wfdBindPipelineToPort`

- `wfdDestroyPipeline`
 - `wfdDestroySource`
 - `wfdBindSourceToPipeline`
 - `wfdGetPipelineAttrib{iv/fv}`
 - `wfdSetPipelineAttrib{i/v/iv/fv}`
- > When flipping to display, you should use a minimum of two `WFDSource` handles (double buffering) to prevent tearing or corruption.
- For more details, see section 8.1 in the [OpenWF Display Specification](#).
- > The following prerequisites must be fulfilled for use cases to suspend to RAM (SC7 mode):
- Before entering suspend-to-RAM (SC7) mode, it is recommended to perform a null flip to display.
 - To exit SC7 mode, a `modeset` must be performed before flipping to display.
- > On Linux platforms, the following restriction applies when you invoke `wfdDeviceCommit` or `wfdDeviceCommitWithNvSciSyncFenceNVX`:
- A main `WFDPipeline` of a `WFDPort` refers to the WFD pipeline that is represented by the first WFD pipeline ID returned by `wfdEnumeratePipelines`. An overlay `WFDPipeline` refers to the WFD pipelines represented by the remaining WFD pipeline IDs enumerated by `wfdEnumeratePipelines`. On Linux, calling the `wfdDeviceCommit` or `wfdDeviceCommitWithNvSciSyncFenceNVX` API involving the overlay WFD pipelines is supported only if the main `WFDPipeline` already has a `WFDSource` bound. Similarly, null flip commit on main WFD pipelines is only supported when all overlay WFD pipelines are also bound to null surfaces.
- > To minimize flip latency, you should set `WFD_PIPELINE_POSTFENCE_SCANOUT_BEGIN_NVX` to `WFD_FALSE`.

6.5.3 Display Serializer

The following sections describe the display serializer.

Configuring Video Timings

For both SST and MST mode, the mode timings that are used for each stream must be configured in Device Tree. Only one mode timing can be specified at a time for each video stream. The timings that are exposed in the EDIDs of the serializer and the panels connected to the downstream deserializer are completely ignored.

An example Device Tree fragment is shown below. In this example, a standard 1920x1080 at 60 Hz timing is specified for the first video stream, and a 1280x720 at 60 Hz timing is specified for the second video stream:

```
\ {
    display@13800000 {
        display-timings {
            display-connector-0 {
                dcb-index = <0>;
```

```

        stream-0 {
            timings-phandle = <&mode0>;
        };

        stream-1 {
            timings-phandle = <&mode1>;
        };
    };

mode0: 1920-1080-60Hz {
    clock-frequency-khz = <148500>;
    hactive = <1920>;
    vactive = <1080>;
    hfront-porch = <88>;
    hback-porch = <148>;
    hsync-len = <44>;
    vfront-porch = <4>;
    vback-porch = <36>;
    vsync-len = <5>;
    rrx1k = <60000>;
    pps-data = [
        11 00 00 89 30 80 04 38
        07 80 04 38 03 c0 03 c0
        02 00 03 58 00 20 73 3e
        00 0d 00 0f 00 1d 00 0e
        18 00 10 f0 03 0c 20 00
        06 0b 0b 33 0e 1c 2a 38
        46 54 62 69 70 77 79 7b
        7d 7e 01 02 01 00 09 40
        09 be 19 fc 19 fa 19 f8
        1a 38 1a 78 22 b6 2a b6
        2a f6 2a f4 43 34 63 74
        00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 ];
};

mode1: 1280-720-60Hz {
    clock-frequency-khz = <74250>;
    hactive = <1280>;
    vactive = <720>;
    hfront-porch = <110>;
    hback-porch = <220>;
    hsync-len = <40>;
    vfront-porch = <5>;
    vback-porch = <20>;
    vsync-len = <5>;
    rrx1k = <60000>;
};
};

```

```
};
```

In the preceding example:

- > "display@13800000" is the overall parent node for the entire display device. This node already exists today.
- "display-timings" is used to specify which timings are used for each stream.
 - "display-connector-0" specifies the timing information for the first display connector. If there are multiple display connectors present on the board that require fixed timings, then a new "display-connector" node must be created for each connector.
 - > "dcb-index" specifies the logical index X of the DCB -> Display Devices -> Display Device X entry in the display DCB blob that this connector entry applies to. If there is only one display connector on the board, then "dcb-index" defaults to 0.
 - > The "stream" nodes specify the phandle of the mode timing node that applies to the given video stream.
- Each "mode" node contains the actual mode timing parameters that will be used for a given video stream.
 - "clock-frequency-khz": Pixel clock frequency in KHz
 - "hactive": Horizontal active
 - "vactive": Vertical active
 - "hfront-porch": Horizontal front porch
 - "hback-porch": Horizontal back porch
 - "hsync-len": Horizontal sync width
 - "vfront-porch": Vertical front porch
 - "vback-porch": Vertical back porch
 - "vsync-len": Vertical sync width
 - "rrx1k": Refresh rate in units of 0.001 Hz
 - "pps-data": All 128B of the DSC PPS

This property should be specified if DSC will be enabled for the given timing.

Each "display-connector" can only have up to two (2) "stream" nodes. Note that it is fine to specify two (2) "stream" nodes even if the display serializer operates in SST mode because only the first "stream" node is consumed by the display driver. The extra node is ignored.

Configuring the Maxim DP Serializer Driver

If you are using the NVIDIA reference Maxim DP serializer driver, there are various ways to configure the driver by using Device Tree. As shown in the following example, a Device Tree fragment configures the Maxim DP serializer chip in MST mode:

```
i2c@31e0000 {
    status = "okay";
    maxim_ser: max_gmsl_dp_ser@40 {
        compatible = "maxim,max_gmsl_dp_ser";
```

```

    reg = <0x40>;
    status = "okay";
    max_gmsl_dp_ser-pwrdn = <&tegra_main_gpio TEGRA234_MAIN_GPIO(G, 3)
GPIO_ACTIVE_HIGH>;
    ser-errb = <&tegra_main_gpio TEGRA234_MAIN_GPIO(G, 7) 0>;
    dprx-link-rate = <0x1e>;
    dprx-lane-count = <0x4>;
    enable-mst;
    mst-payload-ids = <0x1 0x3 0x2 0x4>;
    gmsl-stream-ids = <0x0 0x1 0x2 0x3>;
    gmsl-link-select = <0x0 0x0 0x1 0x1>;
    enable-dp-fec;
    enable-dsc = <0 1>;
    enable-gmsl-fec = <0 1>;
    enable-gmsl3 = <0 1>;

};

};


```

A description of each of the preceding properties:

> Required properties:

- **compatible**: Must be "maxim,max_gmsl_dp_ser".
- **reg**: I2C address of the Maxim DP display serializer.
- **max_gmsl_dp_ser-pwrdn**: GPIO pin number of the PWRDN pin. This pin is used to power up the Maxim DP display serializer chip.
- **gmsl-link-select**: This property is an array of four unsigned 8-bit values that determines the GMSL output link to enable for each video pipe X, Y, Z, and U. The possible values for each pipe are:
 - 0x0 (Link A)
 - 0x1 (Link B)
 - 0x2 (Link A + B)

> Optional properties:

- **dprx-link-rate**: Configures the DP link rate of the serializer chip.

The default value is 0x1E (HBR3). The possible values are:

- 0xA (HBR)
- 0x14 (HBR2)
- 0x1E (HBR3)

- **dprx-lane-count**: Configures the DP lane count of the serializer chip.

The default value is 0x4. The possible values are:

- 0x1
- 0x2
- 0x4

- **ser-errb**: GPIO pin number of the ERRB pin. This pin is used for error and fault reporting by the serializer chip.

- **enable-mst**: This is a Boolean property. If this property is present, the driver will enable MST mode.
- **mst-payload-ids**: This property is an array of four unsigned 8-bit values, which represent MST payload IDs of pipe X, Y, Z, U. This property is mandatory if the enable-mst property is mentioned in dt.
- **gmsl-stream-ids**: This property is an array of four unsigned 8-bit values, which represent GMSL stream IDs of pipe X, Y, Z, U. This property is mandatory if enable-mst property is mentioned in dt.
- **enable-dp-fec**: This is a Boolean property. When this property is present, the driver will enable FEC on the DP link if the serializer chip supports it.
- **enable-dsc**: This property is an array of two 32-bit values, where each value indicates whether DSC is enabled or not. The first entry corresponds to video pipe X, and the second entry corresponds to video pipe Y. DSC is only supported on pipe X currently.
- **enable-gmsl-fec**: This property is an array of two 32-bit values, where each value indicates whether FEC is enabled on the GMSL link. The first entry corresponds to GMSL Link A, and the second entry corresponds to GMSL Link B.
- **enable-gmsl3**: This is a boolean property. When this property is set, GMSL3 capabilities are enabled.

Modeset Limitations

The Maxim DP display serializers do not support dynamic mode changes in MST mode without requiring a reset of the serializer chip in-between. In this context, a mode change refers to changing the number of video streams and/or the display timings that are used for each stream. As such, if the Maxim DP display serializers are configured in MST mode, it is recommended to always enable all video streams that will be used at once.

Configuring the Maxim HDMI Serializer

If you are using the NVIDIA Maxim HDMI serializer driver, there are various ways to configure the driver by using Device Tree, As shown in the following example:

```
i2c@31e0000 {
    status = "okay";
    maxim_ser: maxim_gmsl_hdmi_ser@40 {
        compatible = "maxim,maxim_gmsl_hdmi_ser";
        reg = <0x40>;
        status = "okay";
        maxim_gmsl_hdmi_ser-pwrdsn = <&tegra_main_gpio TEGRA234_MAIN_GPIO(G, 3>;
        GPIO_ACTIVE_HIGH>;
        ser-errb = <&tegra_main_gpio TEGRA234_MAIN_GPIO(G, 7)>;
        enable_rclkout = <0x1>;
    };
};
```

A description of each of the preceding properties:

- > Required properties

- **compatible:** Must be `maxim,maxim_gmsl_hdmi_ser`.
 - **reg:** I2C register address of Maxim HDMI serializer.
 - **maxim_gmsl_hdmi_ser-pwrdsn:** GPIO pin number of the PWRDN pin. This pin is used to power up the Maxim HDMI display serializer chip.
 - **ser-errb:** GPIO pin number of the ERRB pin. This pin is used for error and fault reporting by the serializer chip.
- > Optional properties
- **enable_rclkout:** RCLOUT signal is going as an input to camera deserializer and working as a clock source.

The Maxim HDMI display serializer does not support multiple links and by default uses the Link A of the serializer.

Configuring the TI Serializer Driver

If you use the NVIDIA reference TI serializer driver, there are several ways to configure the driver by using the device tree. The following example shows a device tree fragment that configures the TI serializer chip in SST mode:

```
i2c@31e0000 { /* i2c8 */
    status = "okay";
    ti_ser: ti_fpdlink_dp_ser@18 {
        compatible = "ti,ti_fpdlink_dp_ser";
        reg = <0x18>;
        status = "okay";
        ti_fpdlink_dp_ser-pwrdsn = <&tegra_main_gpio TEGRA234_MAIN_GPIO(G, 3)
        GPIO_ACTIVE_HIGH>;
        dprx-link-rate = <0x0A>;
        dprx-lane-count = <0x4>;
        timings-phandle = <&mode0>;
        fpd-link-select = <0x1 0x0>;
    };
}
```

The properties of the preceding example are described as follows:

- > Required properties:
- **compatible:** Must be `ti,ti_fpdlink_dp_ser`.
 - **reg:** I2C address of the TI display serializer.
 - **ti_fpdlink_dp_ser-pwrdsn:** GPIO pin number of the PWRDN pin. This pin is used to power up the TI display serializer chip.
 - **timings-phandle:** This property should be set as `mode0` or `mode1` to reflect raster or display timings used by the NVIDIA reference platforms.
 - **fpd-link-select:** This property is an array of two unsigned 8-bit values that represent FPDlink port IDs. First field is for link A and the second is for link B. Value 0 indicates that the link is disabled and value 1 indicates that the link is enabled. At any given time, only one link is supported.
- > Optional properties:
- **dprx-link-rate:** Configures the DP link rate of the serializer chip.

The default value is `0x1E` (HBR3). The possible values are:

- 0xA (HBR)
- 0x14 (HBR2)
- 0x1E (HBR3)
- **dprx-lane-count:** Configures the DP lane count of the serializer chip.

The default value is 0x4. The possible values are:

- 0x1
- 0x2
- 0x4

To enable TI Maxim DP Serializer driver configuration on Nvidia Reference Platform p3710, the bind partition command should be executed as follows:

```
bind_partitions -b p3710-12-a01 linux DISP_SER_MODULE=TI983
```

6.5.4 Head to Window Assignment

The following sections describe head to window assignment.

Configuring Head to Window Assignment

Configure head to window assignment in the Device Tree. If an assignment is not specified in the DT, the driver assigns windows (2N) and (2N + 1) to HEAD N. In DT, specify the assignment using 64 bit mask, which is interpreted as:

Head-Bitmask	Window-Number
BITMASK(0-7)	0
BITMASK(8-15)	1
BITMASK(16-23)	2
BITMASK(24-31)	3
BITMASK(32-39)	4
BITMASK(40-47)	5
BITMASK(48-55)	6
BITMASK(56-63)	7

The display driver fails to load if an invalid assignment is specified in the DT. The specified assignment must adhere to the conditions below:

1. The specified window number must be supported by hardware.
2. The specified head number must be supported by hardware.

3. The same window must not be assigned simultaneously for multiple heads.
4. At least one window must be assigned to at least one head (that is, specified window-head mask should not be 0).

The display driver culls the head with no windows assigned, and all the heads above it. For example, if hardware supports three heads and the user uses the assignment mask to assign valid windows to head-0 and head-2 but no windows to head-1, then head-1 and head-2 is culled.

An example Device Tree fragment is shown below. In this example, window-0, window-1, and window-2 is assigned to head-0 and window-3 is assigned to head-1.

```
\ {
    display@13800000 {
        nvidia,window-head-mask = <0x00000000 0x02010101>;
    };
}
```

In the preceding example:

- > `display@13800000` is the overall parent node for the entire display device. This node already exists today.
 - `nvidia,window-head-mask` is used to specify the 64-bit window head assignment mask.

6.5.5 Restrictions

NvDisplay architecture has the following restrictions:

- > Only one NvDisplay client process can be active at a time on the NVIDIA DRIVE Orin™ platforms.
 - On the NVIDIA DRIVE AGX Orin™ Platforms, `nvidia-drm` should not be installed when you run non-DRM applications.
- > NVIDIA DRIVE® OS does not support passive DP-to-HDMI (DP++) cables, dongles, or adapters.

6.5.6 Display State Manager

Drive Setmode

When driver setmode is enabled, display setup and modeset will happen during the `resmgr init` phase instead of deferring this to when the first display client comes up. So, enabling driver setmode can improve "power-on to first pixel visible on screen" latency. Driver setmode can be enabled only for the configuration, which uses DP Serializer, on other configurations enabling it will fail to load display driver.

An example Device Tree fragment is shown below. In this example, driver setmode is enabled.

```
\ {
```

```

display@13800000 {
    nvidia,driver-setmode;
};

};

```

OpenWFD Modeset

OpenWFD mode setting happens in the following way:

- > When a OpenWFD client first initializes the WFD library, OpenWFD queries NvKms for the current mode using NVKMS_IOCTL_GET_CURRENT_MODE
 - If current mode is not NULL,
 - OpenWFD exposes the sole active mode to clients and skips invoking the NvKms APIs for mode query and verification.
 - Any client request for modesets (via wfdSetPortMode) will be ignored by the OpenWFD driver, but a message will be logged in slog2info (not an error message).
 - > This enables existing client apps to work as-is without any changes.
 - If current mode is NULL,
 - OpenWFD uses NvKms' mode query and validation IOCTLs to build a list of available modes and then sets a preferred mode out of these modes (this is usually reported by NvKms).
 - At runtime, OpenWFD performs a modeset iff the WFD client invokes wfdSetPortMode followed by a wfdDeviceCommit. This is the legacy way of doing modesets and all current WFD apps are already following this approach.

6.5.7 Enabling HDMI

The NVIDIA Orin™ SoC can support both the DisplayPort (DP) and HDMI connectors, but not sending display output on both connectors at the same time on a given platform. By default, only DisplayPort is enabled for NVIDIA DRIVE AGX Orin™ platform in NVIDIA DRIVE OS 6.0.

To enable HDMI on your reference platforms with HDMI output, the following platform-specific changes are required:

- > [Modifying Pinmux Configuration for the DPAUX HPD Pin](#)
- > [Modifying DCB Blob to Enable HDMI](#)
- > [Enabling HDMI Hot Plug GPIO in the Device Tree](#)

After making all the preceding platform-specific changes to bring up HDMI, you must perform a full build of the image followed by a full flash of the board for the changes to take effect.

Modifying Pinmux Configuration for the DPAUX HPD Pin

By default, the DPAUX HPD pin is set to SFIO mode for the DisplayPort functionality. For HDMI, the DPAUX HPD pin should be set to GPIO mode.

The following example shows that the DPAUX HPD pin is set to GPIO mode depending on your board schematics:

```
dp_aux_ch0_hpd_pm0 {
    nvidia,pins = "dp_aux_ch0_hpd_pm0";
    nvidia,function = "rsvd1";
    nvidia,pull = <TEGRA_PIN_PULL_UP>;
    nvidia,tristate = <TEGRA_PIN_ENABLE>;
    nvidia,enable-input = <TEGRA_PIN_ENABLE>;
    nvidia,io-high-voltage = <TEGRA_PIN_DISABLE>;
    nvidia,lpdr = <TEGRA_PIN_DISABLE>;
};
```

For information on the pinmux configuration for the DPAUX HPD pin on a specific platform, see [Configuring the Pinmux and GPIO](#).

Modifying DCB Blob to Enable HDMI

Display DCB blob has only the DisplayPort connector enabled by default. HDMI connector must also be enabled on the platform with the HDMI output by using the DCB tool as described in [Display Device Tree](#). The following example shows the DCB blob output on reading the DCB blob with HDMI enabled by using the DCB tool.

```
==> Reading DCB blob ==>

#####
# Tegra DCB BLOB #####
#####
# Display Devices #####
Display Devices:::
Display Devices : [0]
Type : [DP]
CCB : [0]
Heads : 0:[Y] 1:[Y]
Sor : [0 ]
DP Lane Count : [4]
DP Link Rate : [8.1GHz]
Connector : [0]
Bus : [0]
Display Devices : [1]
Type : [TMDS]
CCB : [0]
Heads : 0:[Y] 1:[Y]
Sor : [0 ]
HDMI capable : [1]
Connector : [1]
Bus : [0]
#####
# CCB Entries #####
CCB:::
*CCB entries that have both I2C and AUX ports unused (value = 31) are not displayed
CCB Index : 0
I2C Port : [6]
AUX Port : [0]
#####
# Connector entries #####
Connectors:::
Connector Index : 0x0
Type : [DP]
```

```

Hotplug          : A:[Y]
Connector Index : 0x1
Type            : [HDMI]
Hotplug          : A:[Y]

#####
***** #####

```

Enabling HDMI Hot Plug GPIO in the Device Tree

The following example shows a display device tree fragment that contains the HDMI hot plug GPIO DT property as per display device tree bindings, where the HDMI hot plug GPIO pin is from Tegra Main GPIO controller and Port M and Pin 0 with the GPIO_ACTIVE_HIGH GPIO flags set. For more information, refer to GPIO device tree bindings. The actual GPIO pin used for HDMI hot plug and the default state of the GPIO pin should be determined based on the corresponding platform schematics.

```

\{
    display@13800000 {
        os_gpio_hotplug_a = <&tegra_main_gpio TEGRA234_MAIN_GPIO(M, 0)
        GPIO_ACTIVE_HIGH;
    };
};

```

For information on enabling particular HDMI hotplug GPIO pin in PCT files, see the HDMI Configuration section in [AV PCT Input/Output Resource Assignment](#).

6.6 NvStreams

The NVIDIA SDK provides several different libraries (NvMedia, CUDA, OpenGL) for generating and processing various types of images and higher dimensional data. The interfaces used by these libraries were written independently to serve different needs, and each has its own means of representing memory and synchronization resources. There is no direct way to exchange data between them. Furthermore, most of the details of how and when resources used by these libraries are allocated is hidden from the application(s).

The NvStreams libraries serve two primary purposes:

- They allow resources to be allocated up front, with access restrictions defined by the application(s), and with details of their overall requirements well understood. This is vital for safety-critical systems that must ensure the availability of resources and proper encapsulation of information.
- They allow resources to be exchanged between libraries that otherwise do not have knowledge of each other.

This document describes three libraries. NvSciBuf allows applications to allocate and exchange buffers in memory. NvSciSync allows applications to manage synchronization objects that coordinate when sequence of operations begin and end. NvSciStream layers on NvSciBuf and NvSciSync to provide utilities for streaming sequences of data packets between multiple application modules to support a wide variety of use cases. These

libraries also make use of **NvScilpc** for inter-process/partition/system communication. This library is described in a separate chapter.

6.6.1 Comparison with EGL

Developers familiar with the NVIDIA non-safety SDK may have experience with EGL, which also provides objects (EGLImage, EGLSync, and EGLStream) for sharing resources between libraries. NVIDIA continues to support EGL, but EGL is not suitable for the rigorous requirements of a safety-certified system. Some reasons:

- > Resources are allocated using one library. That library has no knowledge that resources will be shared with another library. There is therefore no guarantee that resources will be allocated in a way that meets the requirements of the other library. Mapping them may not be possible.
- > EGL understands only two-dimensional image data. It cannot handle tensors or other non-image sensor data.
- > The EGL interfaces were not designed with safety in mind and have failure modes not allowed in a safety-certified system.

Porting EGL applications to NvStreams requires more than a simple one-to-one replacement of functions. NvStreams requires the application to be more directly involved than EGL in determining resource requirements, allocating resources, and exchanging resources.

6.6.2 NvStreams Libraries

For more information about NvStreams libraries, see:

- > [NvSciBuf](#)
- > [NvSciSync](#)
- > [NvSciStream](#)

6.6.3 Buffer Allocation

Every hardware engine inside NVIDIA hardware can have a different buffer constraint depending on how the buffer is interpreted by the engine. Hence, sharing a buffer across various engines requires that the allocated buffer satisfy the constraints of all engines that will access that buffer. The existing allocation APIs provided by NvMedia or CUDA only consider the constraints of the engines managed by them.

NvSciBuf is a buffer allocation module that can enable applications to allocate a buffer shareable across various hardware engines that are managed by different engine APIs.

6.6.3.1 Memory Buffer Basics

The buffer allocation model of NvSciBuf is summarized as follows:

If two or more hardware engines want to access a common buffer (for example, one engine is writing data into the buffer and the other engine is reading from the buffer), then:

Allocation Model

1. Applications create an attribute list for each accessor.
2. Set the attributes to define the properties of the buffer they intend to create in the respective attribute list.
 - > Applications must set all datatypes (such as Image datatype, etc.) attributes using NvSciBuf/UMD APIs. Applications must also set the NvSciBuf General attributes directly in the attribute list.
 - > For CUDA, applications must set all the required attributes.



Note:

Applications must ensure they set the `NvSciBufGeneralAttrKey_GpuId` attribute on the CUDA side to specify the IDs of all the GPUs that access the buffer.

3. Reconcile these multiple attribute lists. The process of reconciliation guarantees that a common buffer is allocated that satisfies the constraints of all the accessors.
4. Allocate the buffer using the reconciled attribute list. The reconciled attribute list used for allocating the object is associated with the object until the lifetime of the object.
5. Share the buffer with all the accessors.

Types of Buffers

The hardware engine constraints depend on the type of buffer allocated. The different types of buffers supported by NvSciBuf (applications can choose to allocate one of the following types):

- > **RawBuffer**: Raw memory that is used by an application for storing data.
- > **Image**: Memory used to store image data.
- > **ImagePyramid**: Memory used to store ImagePyramid, a group of images arranged in multiple levels, with each level of image scaled to a specific scaling factor.
- > **NvSciBufArray**: Memory used to store a group of units, where each unit represents data of various basic types such as int, float etc.
- > **Tensor**: Memory used to store Tensor data.

Memory Domain Allocation

Applications can choose to allocate the memory from the following domains:

- > System memory
- > Vidmem of a specific dGPU (on the standard build)

Types of Buffer Attributes

The NvSciBuf attribute can be categorized into the following types:

Datatype attributes: Attributes that are specific to one of the buffer types mentioned in the previous section. If the buffer type in the attribute list is one type, then setting the attributes of another type returns an error.

General attributes: Attributes that are not specific to any buffer type and describes the general properties of the buffer. Some of the examples:

- > NvSciBufGeneralAttrKey_Types: Defines the type of the buffer.
- > NvSciBufGeneralAttrKey_NeedCpuAccess: Defines whether the CPU accesses the buffer.
- > NvSciBufGeneralAttrKey_RequiredPerm: Defines the access permissions expected by this buffer accessor.

6.6.3.1.1 NvSciBuf Module

You must open an `NvSciBufModule` before invoking other `NvSciBuf` API. The `NvSciBuf` module is the library's instance created for that application. All `NvSciBuf` resources created within an application are associated with the `NvSciBufModule` of the application.

6.6.3.1.1.1 NvSciBufModule

```
NvSciBufModule module = NULL;
NvSciError err;
err = NvSciBufModuleOpen(&module);
if (err != NvSciError_Success) {
    goto fail;
}
/* ... */
NvSciBufModuleClose(module);
```

6.6.3.1.2 Attribute Lists

`NvSciBuf` attribute lists are categorized into the following types:

- > Unreconciled Attribute List
 - An application can create an unreconciled attribute list and perform get/set operations for each attribute in an unreconciled attribute list.



Note:

The 'set' operation is allowed only once per attribute.

- Both the `NvSciBufAttrListCreate` and `NvSciBufAttrListIpcImportUnreconciled` APIs return an unreconciled attribute list.
- Applications cannot use unreconciled attribute lists to allocate an `NvSciBuf` object.
- > Reconciled Attribute List
 - A reconciled attribute list is the outcome of reconciliation (i.e., merging and validation) of various unreconciled lists that define final layout/allocation properties of the buffer. The application can use this list to allocate an `NvSciBuf`.

object. Refer to the [Reconciliation](#) section below for more details about the reconciliation process.

- A successful call to `NvSciBufAttrListReconcile`, `NvSciBufAttrListIpcImportReconciled`, or `NvSciBufObjGetAttrList` APIs returns a reconciled attribute list.
 - Applications are not allowed to perform set operations on a reconciled attribute list.
- Conflict Attribute List
- An attribute list returned by an unsuccessful reconciliation process of `NvSciBufAttrListReconcile` API is a conflict attribute list.
 - Applications are not allowed to perform get/set operations on a conflict attribute list.
 - Applications can only use conflict attribute lists to dump its content using the `NvSciBufAttrListDump` API.

6.6.3.2 Multi Datatype Attribute Lists

To support the use cases where applications can perceive a buffer with distinct datatypes, NvSciBuf supports either creating attribute lists with multiple datatypes or reconciling attribute lists of distinct datatypes. For example, if an application wants to create a buffer that is perceived as image by one engine and perceived as tensor by another engine, then the application can create an attribute list with `NvSciBufGeneralAttrKey_Types` attribute set to both image and tensor, reconcile the list and allocate the object. Alternatively, the application can create two different attribute lists, one with image attributes and the other with tensor attributes, reconcile both and allocate the object.

6.6.3.2.1 Limitations

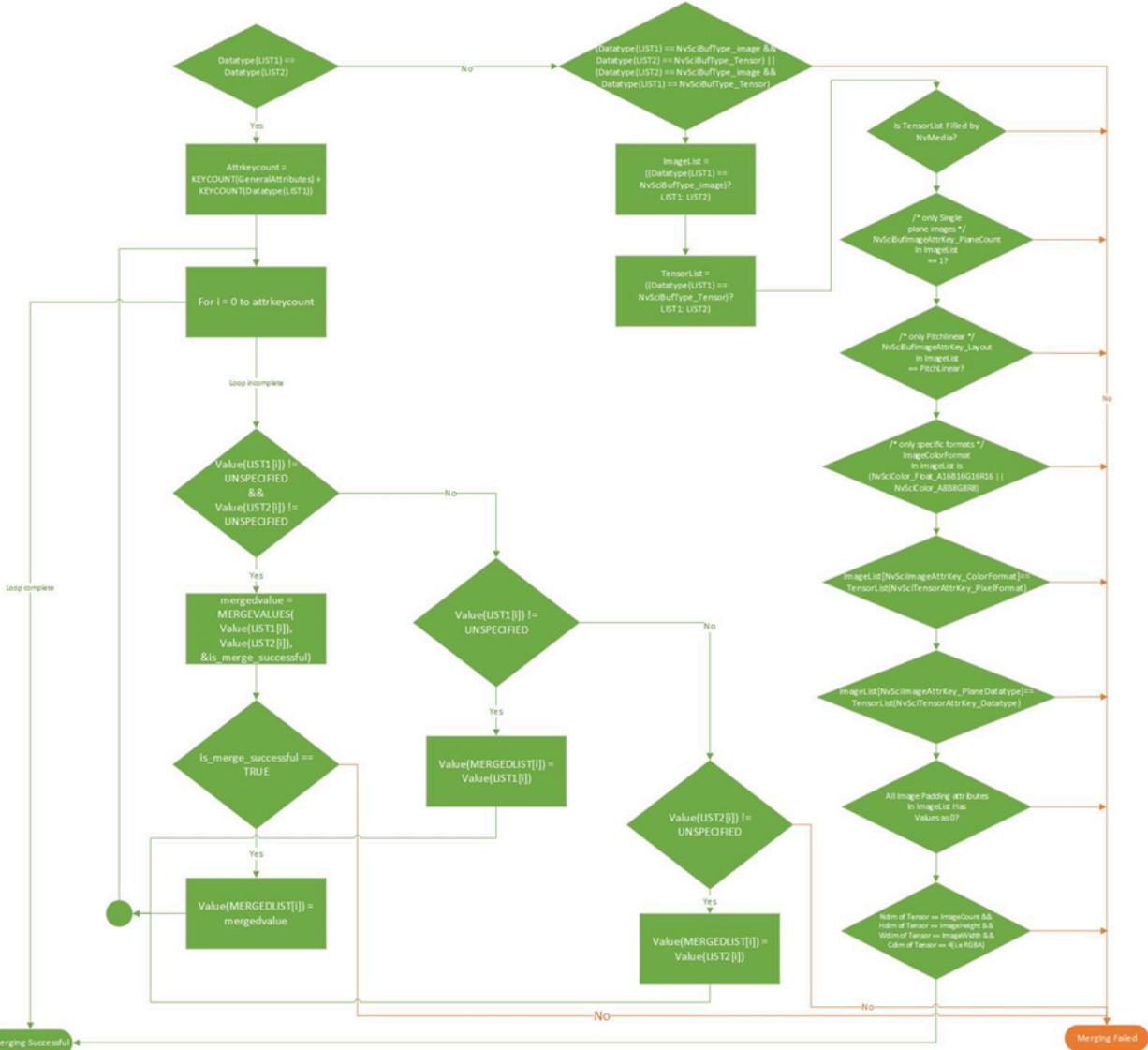
- NvSciBuf supports attribute lists to be either created or reconciled only with image and tensor attributes. Rest of the NvSciBuf datatypes doesn't support co-existence with other datatypes.
- The reconciliation of image & tensor attributes will be successful only if tensor attributes are filled using NvMedia APIs.

6.6.3.3 Reconciliation

An application can initiate the process of reconciliation on one or more attribute lists by invoking `NvSciBufAttrListReconcile` API. The process of NvSciBuf attribute list reconciliation:

1. **Merging:** Values from multiple attribute lists are merged. The process of merging is explained in the flow-chart below.
 - **Datatype(List):** Datatype of the attribute list `list`.
 - **List[i]:** Attribute key named `i` of the attribute list `list`.
 - **Value(List[i]):** Value corresponding to attribute key `i` in the attribute list `list`.
 - **UNSPECIFIED:** Value of an attribute key is ignored and hence unspecified.

- > KEYCOUNT(datatype): Number of attribute keys for the given datatype.
- > MERGEVALUES(value1, value2): This function merges value1 and value2.
 - For most attributes, merging is successful only if value1 equals value2.
 - For attributes like alignment, if value1 is not equal to value2, then the maximum of both is used as the merged value, provided both of them are a power of 2.



2. **Validation:** After merging attributes from multiple lists successfully, validation of merged attributes occurs on the reconciled attribute list, which validates whether all the required attributes are set and the values of all the attributes are valid. For example, after the merging of attributes, if plane-count is set to 2 but the reconciled list contains more values for plane color-format, then validation is unsuccessful.
3. **Output Attributes Computation:** Post validation, output attributes like size, alignment, etc. are computed in the reconciled attribute list.

6.6.3.3.1 NvSciBufAttrLists

```
NvSciBufType bufType = NvSciBufType_RawBuffer;
uint64_t rawsize = (128 * 1024); // Allocate 128K Raw-buffer
uint64_t align = (4 * 1024); // Buffer Alignment of 4K
bool cpuaccess_flag = false;
NvSciBufAttrKeyValuePair rawbuffattrs[] = {
    { NvSciBufGeneralAttrKey_Types, &bufType, sizeof(bufType) },
    { NvSciBufRawBufferAttrKey_Size, &rawsize, sizeof(rawsize) },
    { NvSciBufRawBufferAttrKey_Align, &align, sizeof(align) },
    { NvSciBufGeneralAttrKey_NeedCpuAccess, &cpuaccess_flag,
        sizeof(cpuaccess_flag) },
};
/* Created attrlist1 will be associated with bufmodule */
err = NvSciBufAttrListCreate(bufmodule, &attrlist1);
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciBufAttrListSetAttrs(umd1attrlist, rawbuffattrs,
    sizeof(rawbuffattrs)/sizeof(NvSciBufAttrKeyValuePair));
/*.....*/
NvSciBufAttrListFree(attrlist1);
```

6.6.3.3.2 NvSciBuf Reconciliation

```
NvSciBufAttrList unreconciledList[2] = {NULL};
NvSciBufAttrList reconciledList = NULL;
NvSciBufAttrList ConflictList = NULL;

unreconciledList[0] = AttrList1;
unreconciledList[1] = AttrList2;

/* Reconciliation will be successful if and only all the
 * unreconciledLists belong to same NvSciBufModule and the
 * outputs of this API(i.e either reconciled attribute list
 * or conflict list will also be associated with the same
 * module with which input unreconciled lists belong to.
 */
err = NvSciBufAttrListReconcile(
    unreconciledList,           /* array of unreconciled lists */
    2,                         /* size of this array */
    &reconciledList,           /* output reconciled list */
    &ConflictList);           /* conflict description filled
                                in case of reconciliation failure */

if (err != NvSciError_Success) {
    goto fail;
}
/* ... */
NvSciBufAttrListFree(AttrList1);
NvSciBufAttrListFree(AttrList2);
NvSciBufAttrListFree(reconciledList); //
    In case of successful reconciliation.
NvSciBufAttrListFree(ConflictList); //
```

In case of failed reconciliation.

6.6.3.3.3 Multi Datatype Attribute Lists Reconciliation

```

/* ... */
NvMediaStatus status;
NvMediaTensor *tensor;
NVM_TENSOR_DEFINE_ATTR(tensorAttr);
uint32_t numTensorAttr = NVM_TENSOR_ATTR_MAX;
/* ... */
NvSciBufType bufType = NvSciBufType_Image;
NvSciBufAttrValImageLayoutType layout = NvSciBufImage_PitchLinearType;
uint64_t lrpad = 0, tbpad = 0, imageCount = 1;
bool cpuaccess_flag = true;
bool vpr = false;
uint32_t planecount = 1;
NvSciBufAttrValColorFmt planecolorfmts[] = { NvSciColor_A8B8G8R8 };

NvSciBufAttrValColorStd planecolorstds[] = { NvSciColorStd_SRGB };

NvSciBufAttrValImageScanType planescantype[] = { NvSciBufScan_ProgressiveType };
uint32_t plane_widths[] = { 1920 };
uint32_t plane_heights[] = { 1080 };
NvSciBufAttrKeyValuePair imagebuffattrs[] = {
    { NvSciBufGeneralAttrKey_Types,
        &bufType,
        sizeof(bufType) },
    { NvSciBufGeneralAttrKey_NeedCpuAccess,
        &cpuaccess_flag,
        sizeof(cpuaccess_flag) },
    { NvSciBufImageAttrKey_Layout,
        &layout,
        sizeof(layout) },
    { NvSciBufImageAttrKey_TopPadding,
        &tbpad,
        sizeof(tbpad) },
    { NvSciBufImageAttrKey_BottomPadding,
        &tbpad,
        sizeof(tbpad) },
    { NvSciBufImageAttrKey_LeftPadding,
        &lrpad,
        sizeof(lrpad) },
    { NvSciBufImageAttrKey_RightPadding,
        &lrpad,
        sizeof(lrpad) },
    { NvSciBufImageAttrKey_VprFlag,
        &vpr,
        sizeof(vpr) },
    { NvSciBufImageAttrKey_PlaneCount,
        &planecount,
        sizeof(planecount) },
    { NvSciBufImageAttrKey_PlaneColorFormat,
        planecolorfmts,
        sizeof(planecolorfmts) }
};

```

```

        sizeof(planeColorFmts) },
{ NvSciBufImageAttrKey_PlaneColorStd,
    planeColorStds,
    sizeof(planeColorStds) },
{ NvSciBufImageAttrKey_PlaneWidth,
    plane_widths,
    sizeof(plane_widths) },
{ NvSciBufImageAttrKey_PlaneHeight,
    plane_heights,
    sizeof(plane_heights) },
{ NvSciBufImageAttrKey_ScanType,
    planeScanType,
    sizeof(planeScanType) },
{ NvSciBufImageAttrKey_ImageCount,
    &imageCount,
    sizeof(imageCount) },
};

err = NvSciBufModuleOpen(&bufModule);
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciBufAttrListCreate(bufModule, &app1AttrList);
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciBufAttrListSetAttrs(app1AttrList,
    imageBuffAttrs,
    sizeof(imageBuffAttrs)/sizeof(NvSciBufAttrKeyValuePair));
if (err != NvSciError_Success) {
    goto fail;
}

/* Need to fill NvMediaTensor attributes using Nvmedia APIs */
/* Initialize tensorAttrs as required */
NVM_TENSOR_SET_ATTR_4D(tensorAttr, n, c, h, w, NCHW, INT, 8, UNCACHED, NONE, x);
status = NvMediaTensorFillNvSciBufAttr(NULL, tensorAttr, numTensorAttr, 0,
app2AttrList);

/* .... */
unreconciledLists[0] = app1AttrList;
unreconciledLists[1] = app2AttrList;
err = NvSciBufAttrListReconcileAndObjAlloc(unreconciledLists,
    &nvscibufObj, &conflictList);
if (err != NvSciError_Success) {
    goto fail;
}

/* Create NvMediaTensor from NvSciBufObj */
status = NvMediaTensorCreateFromNvSciBuf(NULL, nvscibufObj, &tensor);
/* .... */

```

6.6.3.4 Buffer Management

6.6.3.4.1 Objects

Applications can use the reconciled attribute list to create any number of NvSciBufObj objects. Each NvSciBufObj represents a buffer and the reconciled attribute list is associated with each object until the object is freed. Applications can create NvMedia/CUDA datatypes out of the allocated buffer using NvMedia/CUDA API and can operate on the buffers by submitting to the NvMedia/CUDA hardware engine using appropriate APIs. Applications wanting to access the buffer from the CPU can set the NvSciBufGeneralAttrKey_NeedCpuAccess attribute to true and get the CPU address using either NvSciBufObjGetCpuPtr or NvSciBufObjGetConstCpuPtr API.



Note:

Invoking NvSciBufObjGetCpuPtr on a read-only buffer or a buffer that does not have CPU access returns an error.

6.6.3.4.1.1 NvSciBuf Object

```
/* Allocate a Buffer using reconciled attribute list and the
 * created NvSciBufObj will be associated with the module to
 * which reconciledAttrlist belongs to.
 */
err = NvSciBufAttrListObjAlloc(reconciledAttrlist,
    &nvscibufobj);
if (err != NvSciError_Success) {
    goto fail;
}
/* .... */
/* Get the associated reconciled attrlist of the object. */
err = NvSciBufObjGetAttrList(nvscibufobj,
    &objReconciledAttrList);
if (err != NvSciError_Success) {
    goto fail;
}
/* .... */
err = NvSciBufObjGetCpuPtr(nvscibufobj, &va_ptr);
if (err != NvSciError_Success) {
    goto fail;
}
/* .... */
NvSciBufAttrListFree(objReconciledAttrList);
NvSciBufObjFree(nvscibufobj);
```

6.6.3.5 VidMem Buffers

NvSciBuf support allocation of buffer from the Vidmem of a specific dGPU. Applications can set the UUID of the specific dGPU to NvSciBufGeneralAttrKey_VidMem_GpuId key and NvSciBuf allocates the buffer from that specific dGPU.


Note:

If one of the unreconciled list sets NvSciBufGeneralAttrKey_VidMem_GpuId key and if any other unreconciled attribute list is filled by NvMedia, then reconciliation of such attribute lists will fail, because NvMedia engines cannot access any dGPU's Vidmem.

6.6.3.5.1 Vidmem Allocation

```

NvSciBufType bufType = NvSciBufType_RawBuffer;
uint64_t rawBufSize = (8U * 1024U);
uint64_t alignment = (4U * 1024U);
bool cpuAccessFlag = false;
NvSciBufAttrKeyValuePair rawBufAttrs[] = {
{
    NvSciBufGeneralAttrKey_Types,
    &bufType,
    sizeof(bufType)
},
{
    NvSciBufRawBufferAttrKey_Size,
    &rawBufSize,
    sizeof(rawBufSize)
},
{
    NvSciBufRawBufferAttrKey_Align,
    &alignment,
    sizeof(alignment)
},
{
    NvSciBufGeneralAttrKey_NeedCpuAccess,
    &cpuAccessFlag,
    sizeof(cpuAccessFlag)
},
{
    NvSciBufGeneralAttrKey_VidMem_GpuId,
    &uuId,
    sizeof(uuId)
}
};
err = NvSciBufModuleOpen(&bufModule);
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciBufAttrListCreate(bufModule, appAttrList);
if (err != NvSciError_Success) {
    goto fail;
}

```

```

        }
        err = NvSciBufAttrListSetAttrs(appAttrList,
rawBufAttrs,
        sizeof(rawBufAttrs)/sizeof(NvSciBufAttrKeyValuePair));
        err = NvSciBufAttrListReconcileAndObjAlloc(appAttrlist,
            &nvscibuf0bj, &conflictList);
        if (err != NvSciError_Success) {
            goto fail;
    }
}

```

6.6.3.6 Inter-Application

If applications involve multiple processes, the exchange of NvSciBuf structures must only go through NvScilpc channels. Each application must open its own NvScilpc endpoint.

6.6.3.6.1 NvScilpc Init

```

NvSciIpcEndpoint ipcEndpoint = 0;
err = NvSciIpcInit();
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciIpcOpenEndpoint("ipc_endpoint", &ipcEndpoint);
if (err != NvSciError_Success) {
    goto fail;
}
/* ... */
NvSciIpcCloseEndpoint(ipcEndpoint);
NvSciIpcDeinit();

```

Applications connected through an NvScilpcEndpoint can exchange NvSciBuf structures (NvSciBufAttrList or NvSciBufObj) using the export/import APIs provided by NvSciBuf.

- NvSciBuf Export APIs return an appropriate export descriptor for the specified NvScilpcEndpoint.
- Applications are responsible for transporting the export descriptor returned by NvSciBuf using the same NvScilpcEndpoint.
- NvSciBuf Import APIs return the respective NvSciBuf structure for the specified export descriptor.
- NvSciBuf provides different APIs for transporting reconciled and unreconciled attribute lists. For reconciled attribute lists, applications can optionally validate the reconciled *list against one or more unreconciled attribute lists* to ensure that the reconciled attribute list satisfies the parameters of the importing process' *unreconciled* lists. This can be done by either passing unreconciled lists to NvSciBufAttrListIpcImportReconciled API while importing or by invoking NvSciBufAttrListValidateReconciled API after importing.

6.6.3.6.2 NvSciBufObj Permissions

- Applications have two options to specify their intended permissions on the NvSciBufObj:

- **Option 1:** Set the intended permissions to NvSciBufGeneralAttrKey_RequiredPerm attribute in the unreconciled attribute-list. If no permissions are specified by the application, NvSciBufAccessPerm_Readonly becomes default value.
 - **Option 2:** Set the intended permissions in the object export/import APIs.
- > The final permissions of the NvSciBufObj for an application is computed by NvSciBuf based on multiple factors explained below:
- Applications that allocates the NvSciBufObj using NvSciBufObjAlloc or NvSciBufAttrListReconcileAndObjAlloc APIs will always get Read/Write permissions.
 - For applications importing the NvSciBufObj, the permissions are computed based the following:

6.6.3.6.3 Notations

- > **ReconList.Perm** – The value of NvSciBufGeneralAttrKey_ActualPerm key in the reconciled attribute list. In simple cases, this value is same as the value of NvSciBufGeneralAttrKey_RequiredPerm key set by the application in its unreconciled attribute list. In cases where an application has multiple unreconciled lists, the NvSciBufGeneralAttrKey_ActualPerm value is the maximum value of the NvSciBufGeneralAttrKey_RequiredPerm key of all the unreconciled lists.
- **ExportAPI.Perm** – The value of export permissions argument used by application with object export APIs such as NvSciBufIpcExportAttrListAndObj or NvSciBufObjIpcExport API.
 - **ImportAPI.Perm** – The value of import permissions argument used by application with object import APIs such as NvSciBufIpcImportAttrListAndObj or NvSciBufObjIpcImport API.
 - **ObjExport.Perm** – The value of final computed permissions with which NvSciBufObj is exported from the export application.
 - **ObjImport.Perm** – The value of final permissions with which NvSciBufObj is imported by the import application.

6.6.3.6.4 Computation of Object Export Permissions

- > **Case-1: ExportAPI.Perm = NvSciBufAccessPerm_Auto.** Export Application invoked the Export API with NvSciBufAccessPerm_Auto option. In this case:

```
ObjExport.Perm = ReconList.Perm
```

- > **Case-2: ExportAPI.Perm = explicit permissions.** Export Application invoked the Export API by explicitly specifying the permissions. In this case, ObjExport.Perm is computed as shown below:

INPUT		OUTPUT
ReconList.Perm	ExportAPI.Perm	ObjExport.Perm
RO	RO	RO

INPUT		OUTPUT
ReconList.Perm	ExportAPI.Perm	ObjExport.Perm
RO	RW	RW
RW	RO	RW
RW	RW	RW

6.6.3.6.5 Computation of Imported NvSciBufObj Permissions

- > **Case-1: ImportAPI.Perm = NvSciBufAccessPerm_Auto.** Import Application invoked the Import API with NvSciBufAccessPerm_Auto option. In this case:

ObjImport.Perm = ObjExport.Perm

- > **Case-2: ImportAPI.Perm = explicit permissions.** Import Application invoked the Import API by explicitly specifying the permissions. In this case, ObjImport.Perm is computed as shown below:

INPUT		OUTPUT
ObjExport.Perm	ImportAPI.Perm	ObjImport.Perm
RO	RO	RO
RO	RW	Import Fail
RW	RO	RW
RW	RW	RW



Note:

While computing the imported NvSciBufObj permissions, the permissions in the reconciled list are ignored. In fact, the value of NvSciBufGeneralAttrKey_ActualPerm in the reconciled list is updated to the value of ObjImport.Perm.



Note:

NvSciBufObj Import APIs return a failure if the export descriptors are imported using a wrong ipcEndpoint or inappropriate permissions.

6.6.3.6.6 Export/Import NvSciBuf AttrLists

```

/* -----App Process1 -----*/
NvSciBufAttrList AttrList1 = NULL;
void* ListDesc = NULL;
size_t ListDescSize = 0U;
/* creation of the attribute list, receiving other lists from other listeners */
err = NvSciBufAttrListIpcExportUnreconciled(
    &AttrList1, /* array of unreconciled lists to be exported */
    1, /* size of the array */
    ipcEndpoint, /* valid and opened NvSciIpcEndpoint intended to
send the descriptor through */
    &ListDesc, /* The descriptor buffer to be allocated and filled
in */
    &ListDescSize ); /* size of the newly created buffer */
if (err != NvSciError_Success) {
    goto fail;
}
/* send the descriptor to the process2 */
/* wait for process 1 to reconcile and export reconciled list */
err = NvSciBufAttrListIpcImportReconciled(
    module, /* NvSciBuf module using which this attrlist to be
imported */
    ipcEndpoint, /* valid and opened NvSciIpcEndpoint on which the
descriptor is received */
    ListDesc, /* The descriptor buffer to be imported */
    ListDescSize, /* size of the descriptor buffer */
    &AttrList1, /* array of unreconciled lists to be used for
validating the reconciled list */
    1, /* Number of unreconciled lists */
    &reconciledAttrList, /* Imported reconciled list */
    if (err != NvSciError_Success) {
        goto fail;
    }
/* -----App Process2 -----*/
void* ListDesc = NULL;
size_t ListDescSize = 0U;
NvSciBufAttrList unreconciledList[2] = {NULL};
NvSciBufAttrList reconciledList = NULL;
NvSciBufAttrList newConflictList = NULL;
NvSciBufAttrList AttrList2 = NULL;
NvSciBufAttrList importedUnreconciledAttrList = NULL;
/* create the local AttrList */
/* receive the descriptor from the other process */
err = NvSciBufAttrListIpcImportUnreconciled(module, ipcEndpoint,
    ListDesc, ListDescSize,
    &importedUnreconciledAttrList);
if (err != NvSciError_Success) {
    goto fail;
}
/* gather all the lists into an array and reconcile */
unreconciledList[0] = AttrList2;
unreconciledList[1] = importedUnreconciledAttrList;
err = NvSciBufAttrListReconcile(unreconciledList, 2, &reconciledList,

```

```

        &newConflictList);
    if (err != NvSciError_Success) {
        goto fail;
    }
    err = NvSciBufAttrListIpcExportReconciled(
        &AttrList1,           /* array of unreconciled lists to be
exported */
        ipcEndpoint,         /* valid and opened NvSciIpcEndpoint
intended to send the descriptor through */
        &ListDesc,           /* The descriptor buffer to be
allocated and filled in */
        &ListDescSize );     /* size of the newly created
buffer */
    if (err != NvSciError_Success) {
        goto fail;
    }
}

```

6.6.3.6.7 Export/Import NvSciBufObj

```

/* process1 */
void* objAndList;
size_t objAndListSize;
err = NvSciBufIpcExportAttrListAndObj(
    bufObj,             /* bufObj to be exported
(the reconciled list is inside it) */
    NvSciBufAccessPerm_ReadOnly, /* permissions we want the
receiver to have */
    ipcEndpoint,         /* IpcEndpoint via which the
object is to be exported */
    &objAndList,          /* descriptor of the object
and list to be communicated */
    &objAndListSize);    /* size of the descriptor */

/* send via Ipc */
/* process2 */
void* objAndList;
size_t objAndListSize;
err = NvSciBufIpcImportAttrListAndObj(
    module,              /* NvSciBufModule use to
create original unreconciled lists in the waiter */
    ipcEndpoint,         /* ipcEndpoint from which the
descriptor was received */
    objAndList,           /* the descriptor of the buf obj
and associated reconciled attribute list received from the signaler */
    objAndListSize,       /* size of the descriptor */
    &AttrList1,            /* the array of original
unreconciled lists prepared in this process */
    1,                  /* size of the array */
    NvSciBufAccessPerm_ReadOnly, /* permissions expected by
this process */
    10000U,               /* timeout in microseconds.
Some primitives might require time to transport all needed resources */
    &bufObj);            /* buf object generated from
the descriptor */

```

```
/* use the buf object */
NvSciBufObjFree(bufObj);
```

6.6.3.6.8 Secure Sharing of NvSciBufObj

NvSciBuf supports secure sharing on x86 with the aid of nvsci_mm and nvsciipc KMD (Kernel Mode Drivers). NvSciBuf will fall back to non-secure sharing of buffers if any of the two KMDs are not installed.

nvsci_mm KMD

Nvsci_mm facilitates secure sharing of NvSciBufObj by mutually authenticating the import using NvScilpcEndpoint as identifier. Without nvsciipc KMD, nvsci_mm cannot perform mutual authentication of the importer process and will fall back to no authentication.

nvsci_mm has three configurable settings that can be adjusted during installation of KMD.

1. Maximum pending exports: During installation of nvsci_mm KMD, you can set an upper limit of the number of pending NvSciBufObj exports. Any export request above the maximum configured limit will fail.

This setting protects against a malicious process trying to export a NvSciBufObj multiple times, overwhelming the nvsci_mm KMD, and affecting all kernel system resources.

2. User group authorization: During installation of nvsci_mm KMD, you can restrict access to nvsci_mm device node /dev/nvsci_mm to certain Linux users within the allowed Linux user group. Applications run by Linux users that are part of the allowed Linux user group can import and export buffers. Applications run by the Linux users that are not part of the allowed Linux user group will fail to import or export buffers.
3. Mutual authentication of importer: This feature is active if nvsciipc KMD is installed. During export, nvsci_mm stores the authentication data of the NvScilpcEndpoint, which is paired to the exporter's NvScilpcEndpoint. During import, nvsci_mm verifies authentication, and then importing of buffers is allowed. This ensures that the buffer can be imported by the NvScilpcEndpoint to which the exporter intended to export.

6.6.3.7 NvSciBuf API

For information about NvSciBuf API, see [Buffer Allocation APIs](#).

6.6.3.8 UMD Access

6.6.3.8.1 cuDLA

cuDLA supports importing an NvSciBufObj into DLA as cuDLA external memory of type NvSciBuf. Users can use cuDLA APIs to import memory objects and get the pointer to be passed while submitting DLA tasks. If the NvSciBuf object is imported into DLA using cuDLA and also mapped by other drivers, then the application must use cuDLA external

semaphore APIs described in this section as appropriate barriers to maintain coherence between cuDLA and the other drivers.

6.6.3.8.1.1 Importing a NvSciBufObj into cuDLA

Following are the steps required to use NvSciBufObj as a pointer.

1. Allocate NvSciBufObj.

The application creates NvSciBufAttrList. If the same object is used by other UMDs, the corresponding attribute lists must be created and reconciled. The reconciled list must be used to allocate the NvSciBufObj.



Note: The attribute list and NvSciBuf objects must be maintained by the application.

2. Query NvSciBufObj attributes to fill cuDLA descriptors.

The application must query the allocated NvSciBufObj for required attributes to fill the cuDLA external memory descriptor.

3. NvSciBuf objects registration with cuDLA.

`cudlaImportExternalMemory()` must be used to register the allocated NvSciBuf object with DLA by filling up `cudlaExternalMemoryHandleDesc`. This API will return a pointer that can be used to submit cuDLA tasks.

4. Dereistration of NvSciBuf objects from cuDLA.

`cudlaMemUnregister()` API must be used to unregister the NvSciBuf from DLA.

6.6.3.8.2 2D/DLA/LDC/SIPL/Multimedia

2D/DLA/LDC/SIPL/Multimedia supports different datatypes, such as Image, Tensor, Array, etc. Each provides a set of interfaces for each data type to interact with NvSciBuf.

6.6.3.8.2.1 2D/LDC/SIPL/Multimedia and NvSciBuf Interaction

This section describes 2D/LDC/SIPL/Multimedia and NvSciBuf interaction. The following steps show the typical flow to allocate and register an NvSciBuf object:

1. The application creates a NvSciBufAttrList.
2. The application queries 2D/LDC/SIPL/Multimedia to fill in the NvSciBufAttrList by calling the relevant APIs (`NvMedia2DFillNvSciBufAttrList/`
`NvMediaLdcFillNvSciBufAttrList/`
`INvSIPLCamera::GetImageAttributes/`
`NvMediaEPFillNvSciBufAttrList/`
`NvMediaJPDFillNvSciBufAttrList/`
`NvMediaJPEFillNvSciBufAttrList/`
`NvMediaOFAFillNvSciBufAttrList/`
`NvMediaDEFillNvSciBufAttrList/etc.`) on the NvSciBufAttrList
3. The application may choose to set any of the public NvSciBuf attributes, which are not set by those APIs.
4. If the same NvSciBuf object is shared with other UMDs, then the application can get the corresponding NvSciBufAttrList from the respective UMD.
5. The application asks NvSciBuf to reconcile all the filled NvSciBufAttrLists and then allocates an NvSciBuf object.
6. The allocated NvSciBufObj is then registered with 2D/LDC/SIPL/Multimedia

6.6.3.8.3 CUDA

CUDA supports the import of an NvSciBufObj into CUDA as CUDA external memory of type NvSciBuf. Once imported, use CUDA API to get a CUDA pointer/array from the imported memory object, which can be passed to CUDA kernels. Applications must query NvSciBufObj for the attributes required to fill descriptors, which are passed as parameters to the import/map APIs.

If the NvSciBuf object imported into CUDA is also mapped by other drivers, then the application must use CUDA external semaphore APIs described here as appropriate barriers to maintain coherence between CUDA and the other drivers.

6.6.3.8.3.1 NvSciBufObj as a CUDA Pointer / Array

The following section describes the steps required to use NvSciBufObj as a CUDA pointer/array.

1. Allocate NvSciBufObj.
 - > The application creates NvSciBufAttrList and sets the `NvSciBufGeneralAttrKey_GpuId` attribute to specify the ID of the GPU that shares the buffer, along with other attributes.
 - > If the same object is used by other UMDs, the corresponding attribute lists must be created and reconciled. The reconciled list must be used to allocate the NvSciBufObj.



Note:

The attribute list and NvSciBuf objects must be maintained by the application.

2. Query NvSciBufObj attributes to fill CUDA descriptors.
 - > The application must query the allocated NvSciBufObj for required attributes to fill the CUDA external memory descriptor.
3. NvSciBuf object registration with CUDA.
 - > `cudaImportExternalMemory()` must be used to register the allocated NvSciBuf object with CUDA by filling up `cudaExternalMemoryHandleDesc` for type `cudaExternalMemoryHandleTypeNvSciBuf`.
 - > `cudaDestroyExternalMemory()` API must be used to free the CUDA external memory. CUDA mappings created from external memory must be freed before invoking this API.
4. Getting CUDA pointer/array from imported external memory.
 - > `cudaExternalMemoryGetMappedBuffer()` maps a buffer onto an imported memory object and returns a CUDA device pointer. The properties of the buffer must be described in the CUDA ExternalMemory buffer description by querying attributes from NvSciBufObj. The returned pointer device pointer must be freed using `cudaFree`.
 - > `cudaExternalMemoryGetMappedMipmappedArray()` maps a CUDA mipmapped array onto an external object and returns a handle to it. The properties of the buffer must be described in the CUDA ExternalMemory MipmappedArray desc

by querying attributes from NvSciBufObj. The returned CUDA mipmapped array must be freed using cudaFreeMipmappedArray.


Note:

All the APIs mentioned in the sections above are CUDA-runtime APIs. Each of them has an equivalent driver API. The syntax and usage of both versions are the same.

6.6.3.8.3.2 NvSciBuf-CUDA Interop

```
***** Allocate NvSciBuf object *****/
// Raw Buffer Attributes for CUDA
NvSciBufType bufType = NvSciBufType_RawBuffer;
uint64_t rawsize = SIZE;
uint64_t align = 0;
bool cpuaccess_flag = true;
NvSciBufAttrValAccessPerm perm = NvSciBufAccessPerm_ReadWrite;
uint64_t gpuId[] = {};
cuDeviceGetUuid(&uuid, dev));
gpuId[0] = (uint64_t)uuid.bytes;
// Fill in values
NvSciBufAttrKeyValuePair rawbuffattrs[] = {
    { NvSciBufGeneralAttrKey_Types, &bufType, sizeof(bufType) },
    { NvSciBufRawBufferAttrKey_Size, &rawsize, sizeof(rawsize) },
    { NvSciBufRawBufferAttrKey_Align, &align, sizeof(align) },
    { NvSciBufGeneralAttrKey_NeedCpuAccess, &cpuaccess_flag,
        sizeof(cpuaccess_flag) },
    { NvSciBufGeneralAttrKey_RequiredPerm, &perm, sizeof(perm) },
    { NvSciBufGeneralAttrKey_GpuId, &gpuId, sizeof(gpuId) },

};

// Create list by setting attributes
err = NvSciBufAttrListSetAttrs(attrListBuffer, rawbuffattrs,
    sizeof(rawbuffattrs)/sizeof(NvSciBufAttrKeyValuePair));
NvSciBufAttrListCreate(NvSciBufModule, &attrListBuffer);
// Reconcile And Allocate
NvSciBufAttrListReconcile(&attrListBuffer, 1, &attrListReconciledBuffer,
&attrListConflictBuffer)
NvSciBufObjAlloc(attrListReconciledBuffer, &bufferObjRaw);
***** Query NvSciBuf Object *****/
NvSciBufAttrKeyValuePair bufattrs[] = {
    {NvSciBufRawBufferAttrKey_Size, NULL, 0},
};

NvSciBufAttrListGetAttrs(retList, bufattrs, sizeof(bufattrs)/
sizeof(NvSciBufAttrKeyValuePair)));
ret_size = *(static_cast<const uint64_t*>(bufattrs[0].value));
***** NvSciBuf Registration With CUDA *****/
// Fill up CUDA_EXTERNAL_MEMORY_HANDLE_DESC
cudaExternalMemoryHandleDesc memHandleDesc;
memset(&memHandleDesc, 0, sizeof(memHandleDesc));
memHandleDesc.type = cudaExternalMemoryHandleTypeNvSciBuf;
memHandleDesc.handle.nvSciBufObject = bufferObjRaw;
memHandleDesc.size = ret_size;
```

```

cudaImportExternalMemory(&extMemBuffer, &memHandleDesc);
/********************* Mapping to CUDA *****/
cudaExternalMemoryBufferDesc bufferDesc;
memset(&bufferDesc, 0, sizeof(bufferDesc));
bufferDesc.offset = offset = 0;
bufferDesc.size = ret_size;
cudaExternalMemoryGetMappedBuffer(&dptra, extMemBuffer, &bufferDesc);
/******************* CUDA Kernel *****/
// Run CUDA Kernel on dptra
/******************* Free CUDA mappings *****/
cudaFree();
cudaDestroyExternalMemory(extMemBuffer);
/******************* Free NvSciBuf *****/
NvSciBufAttrListFree(attrListBuffer);
NvSciBufAttrListFree(attrListReconciledBuffer)
NvSciBufAttrListFree(attrListConflictBuffer);
NvSciBufObjFree(bufferObjRaw);

```

6.6.3.8.4 Vulcan SC

6.6.3.8.4.1 Import NvSciBuf to Vulkan SC Resource Types

Vulkan SC supports the import of NvSciBufObj into VkDeviceMemory as Vulkan SC external memory of type NvSciBuf. Once imported, the VkDeviceMemory can be used as a normal Vulkan SC memory object. When binding to other Vulkan SC resources, such as VkBuffer and VkImage, the applications must ensure the attributes of NvSciBufObj match the bound Vulkan SC resources. Once the NvSciBufObj is imported to VkBuffer and VkImage, the applications can use them in Vulkan SC applications as normal resource types.

If the NvSciBufObj imported to Vulkan SC is also mapped by other drivers, or to a separate VkDevice, the applications must use NvSciSync APIs and the Vulkan SC VK_NV_external_sci_sync2 extension to maintain the correct dependencies between Vulkan SC and other drivers.

The following section describes the steps to allocate NvSciBufObj for Vulkan SC resources (VkBuffer and VkImage), and the steps to import to VkDeviceMemory.

Allocate NvSciBufObj for VkBuffer

- > Create a VkBuffer object.
 - Use the vkCreateBuffer API to create a VkBuffer. This API takes VkBufferCreateInfo as a parameter; the application must chain VkExternalMemoryBufferCreateInfo with VkExternalMemoryBufferCreateInfo::handleTypes of VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV to the VkBufferCreateInfo::pNext.
- > Allocate an NvSciBufObj.
 - Use vkGetBufferMemoryRequirements API to get the memory requirement of the created VkBuffer, then fill the required attributes of the NvSciBufAttrList. For VkBuffer, the NvSciBufType field is always NvSciBufType_RawBuffer.

- Use `vkGetPhysicalDeviceSciBufAttributesNV` to set the `NvSciBufGeneralAttrKey_GpuId` attribute to specify the ID of the GPU that shares the buffer, along with other attributes.
- Allocate `NvSciBufObj` by using the `NvSciBuf` API.

Allocate NvSciBufObj for VkImage

- > Create a `VkImage` object.
 - Use the `vkCreateImage` API to create a `VkImage`. This API takes `VkImageCreateInfo` as a parameter. The application must chain a `VkExternalMemoryImageCreateInfo` struct with `VkExternalMemoryImageCreateInfo::handleTypes` of `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV`.
- > Allocate an `NvSciBufObj`
 - Use the `vkGetImageMemoryRequirements` API to get the memory requirement of created `VkImage`, and fill the required attribute of `NvSciBufAttrList`. For `VkImage`, the `NvSciBufType` field is always `NvSciBufType_Image`.
 - Use `vkGetPhysicalDeviceSciBufAttributesNV` to set the `NvSciBufGeneralAttrKey_GpuId` attribute to specify the ID of the GPU that shares the buffer, along with other attributes.
 - Allocate `NvSciBufObj` by using the `NvSciBuf` API.
- > Attributes mapping between `VkImage` and `NvSciBufObj`.
 - Vulkan SC only supports two image formats for imported `NvSciBufObj`: `NvSciColor_A8R8G8B8`, and `NvSciColor_A8B8G8R8`. The matching formats on Vulkan SC are `NvSciColor_A8R8G8B8 => VK_FORMAT_B8G8R8A8_UNORM`, and `NvSciColor_A8B8G8R8 => VK_FORMAT_A8B8G8R8_UNORM_PACK32`.
 - Vulkan SC supports both `NvSciBufImage_BlockLinearType` and `NvSciBufImage_PitchLinearType` memory layouts. The matching layouts on Vulkan SC are `NvSciBufImage_PitchLinearType => VK_IMAGE_TILING_LINEAR`, and `NvSciBufImage_BlockLinearType => VK_IMAGE_TILING_OPTIMAL`.
 -  **Note:** As Vulkan SC and `NvSciBufObj` use different ways to calculate the pitch size in memory layout of pitch linear, it is possible that for some image width, the pitch size will be different between Vulkan SC and `NvSciBuf`. Vulkan SC will report `VK_ERROR_VALIDATION_FAILED` during `vkBindImageMemory` in such a case. The recommendation is to use `NvSciBufImage_BlockLinearType` to avoid such issues and improve the performance of image usage.

Import NvSciBufObj to VkDeviceMemory

- > Use the memory requirement information queried in the previous sections to fill the `VkMemoryAllocateInfo` struct. Use the `vkAllocateMemory` API to create a `VkDeviceMemory`. This API accepts `VkMemoryAllocateInfo` as an input parameter, and applications must chain a valid `VkImportMemorySciBufInfoNV` struct to `VkMemoryAllocateInfo::pNext` with `VkImportMemorySciBufInfoNV::handleType` of `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV` and `VkImportMemorySciBufInfoNV::handle` of allocated `NvSciBufObj`.

Bind VkDeviceMemory to Vulkan SC resources

- Use the `vkBindBufferMemory` API to bind `VkDeviceMemory` to `VkBuffer`.
- Use the `vkBindImageMemory` API to bind `VkDeviceMemory` to `VkImage`.

6.6.3.8.4.2 Export NvSciBuf from Vulkan SC

In the Vulkan SC Specification, `VK_NV_external_memory_sci_buf` also defines a feature to export an `NvSciBufObj`. In Vulkan SC terminology, “**export**” indicates the Vulkan SC driver allocates and manages the `NvSciBufObj` instead of the application. However, to keep the interfaces symmetrical with other DriveOS UMDs, this feature is always disabled. Applications can use the `vkGetPhysicalDeviceFeatures2` API by passing a `VkPhysicalDeviceExternalMemorySciBufFeaturesNV` struct to query the enablement of this feature.

6.6.3.8.4.3 Vulkan SC-NvSciBuf Interop Examples

This section provides example code to demonstrate practical usage of `VK_NV_external_memory_sci_buf`. The example is edited for brevity, to focus on the interoperability of Vulkan SC and NvSciBuf, and is incomplete for the complete Vulkan SC creation. Your application must check each `VkResult` and `NvSciError`, and API function returns, and handle error values.

VkBuffer - NvSciBuf

```
// Create VkBuffer
VkBuffer vkbuffer = VK_NULL_HANDLE;
const uint32_t queueFamilyIndex = 0;
VkExternalMemoryBufferCreateInfo externalMemoryBufferInfo = {
    .sType = VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO,
    .pNext = nullptr,
    .handleTypes = VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV
};
VkDeviceSize size = 256;
VkBufferCreateInfo bufferInfo = {
    .sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO,
    .pNext = &externalMemoryBufferInfo,
    .flags = 0,
    .size = size,
    .usage = VK_BUFFER_USAGE_TRANSFER_DST_BIT,
    .sharingMode = VK_SHARING_MODE_EXCLUSIVE,
    .queueFamilyIndexCount = 1,
    .pQueueFamilyIndices = &queueFamilyIndex,
};
vkCreateBuffer(m_dev, &bufferInfo, nullptr, &vkbuffer);

VkMemoryRequirements memReqs = {};
vkGetBufferMemoryRequirements(dev, vkbuffer, &memReqs);
uint64_t alignment = memReqs.alignment;
bool needCpuAccess = false;
bool needCpuCached = false;
NvSciBufType bufType = NvSciBufType_RawBuffer;
NvSciBufAttrValAccessPerm perm = NvSciBufAccessPerm_ReadWrite;
```

```

needCpuAccess = (memReqs.memoryTypeBits & VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
needCpuCached = (memReqs.memoryTypeBits & VK_MEMORY_PROPERTY_HOST_CACHED_BIT);

NvSciBufAttrKeyValuePair pairArray[] = {
{ NvSciBufRawBufferAttrKey_Size, (void*) &memReqs.size, sizeof(memReqs.size) },
{ NvSciBufRawBufferAttrKey_Align, (void*) &alignment, sizeof(alignment) },
{ NvSciBufGeneralAttrKey_Types, (void*) &bufType, sizeof(bufType) },
{ NvSciBufGeneralAttrKey_NeedCpuAccess, (void*) &needCpuAccess, sizeof(needCpuAccess) },
{ NvSciBufGeneralAttrKey_EnableCpuCache, (void*) &needCpuCached,
  sizeof(needCpuCached) },
{ NvSciBufGeneralAttrKey_RequiredPerm, (void*) &perm, sizeof(perm) },
};

NvSciBufAttrList attrList, reconciledList, conflictList;
NvSciBufAttrListCreate(scibufModule, &attrList)

// Application fills the public key-value pairs
NvSciBufAttrListSetAttrs(attrList, pairArray,           sizeof(pairArray)/
sizeof(NvSciBufAttrKeyValuePair));

// Driver will fill the NvSciBufGeneralAttrKey_GpuId
vkGetPhysicalDeviceSciBufAttributesNV(m_physDev, attrList);

// Allocate NvSciBufObj
NvSciBufAttrListReconcile(&attrList, 1, &reconciledList, &conflictList);
NvSciBufObj sciBufObjVkBuffer {nullptr};
NvSciBufObjAlloc(reconciledList, &sciBufObjVkBuffer);

// Create VkDeviceMemory
VkImportMemorySciBufInfoNV importSciBufInfo {
    .sType = VK_STRUCTURE_TYPE_IMPORT_MEMORY_SCI_BUF_INFO_NV;
    .pNext = nullptr;
    .handleType = VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV;
    .handle = sciBufObjVkBuffer;
};

// GetMemoryType is a helper function to convert memoryTypeBits to memoryType index,
// skip the detail of implementation here.
VkMemoryAllocateInfo memAllocInfo = {
    .sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
    .pNext = importSciBufInfo;
    .memoryTypeIndex = needCpuAccess ?
        GetMemoryType(memReqs.memoryTypeBits, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT):
        GetMemoryType(memReqs.memoryTypeBits, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
    .allocationSize = 0; // ignore the allocate size when importing NvSciBufObj
};

VkDeviceMemory devMemory;
vkAllocateMemory(dev, &memAllocInfo, NULL, &devMemory);
vkBindBufferMemory(dev, vkbuffer, devMemory);

// De-init
vkDestroyBuffer(dev, vkbuffer, 0);
NvSciBufAttrListFree(attrList);
NvSciBufAttrListFree(reconciledList);

```

```
NvSciBufAttrListFree(conflictList);
NvSciBufObjFree(sciBufObjVkBuffer);
```

VkImage - NvSciBuf

```
// Create VkImage
VkExternalMemoryImageCreateInfo externalMemInfo = {
    .sType = VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO,
    .pNext = nullptr,
    .handleTypes = VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV
};

VkImageCreateInfo imageInfo = {
    .sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO,
    .pNext = &externalMemInfo,
    .flags = 0,
    .imageType = VK_IMAGE_TYPE_2D,
    .format = VK_FORMAT_B8G8R8A8_UNORM,
    .extent = {128, 128, 1},
    .mipLevels = 1,
    .arrayLayers = 1,
    .samples = VK_SAMPLE_COUNT_1_BIT,
    .tiling = VK_IMAGE_TILING_LINEAR,
    .usage = VK_IMAGE_USAGE_TRANSFER_SRC_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT,
    .sharingMode = VK_SHARING_MODE_EXCLUSIVE,
    .queueFamilyIndexCount = 0,
    .pQueueFamilyIndices = nullptr,
    .initialLayout = VK_IMAGE_LAYOUT_UNDEFINED
};
VkImage vkimage = VK_NULL_HANDLE;
vkCreateImage(m_dev, &imageInfo, NULL, &vkimage);

VkMemoryRequirements* memReqs
vkGetImageMemoryRequirements(m_dev, vkimage, memReqs);

// Fill the NvSciBufAttrList based on VkImage's attributes
NvSciBufType bufType = NvSciBufType_Image;
NvSciBufAttrValAccessPerm perm = NvSciBufAccessPerm_ReadWrite;
NvSciBufAttrValImageLayoutType layout = NvSciBufImage_BlockLinearType;
NvSciBufAttrValImageScanType planescantype = NvSciBufScan_ProgressiveType;
uint32_t planeCount = 1;
NvSciBufAttrValColorFmt colorFormat = NvSciColor_A8R8G8B8;
uint32_t height = imageInfo.extent.height;
uint32_t width = imageInfo.extent.width;
bool needCpuAccess = (memReqs.memoryTypeBits & VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
bool needCpuCached = false;

needCpuAccess = (memReqs.memoryTypeBits & VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
needCpuCached = (memReqs.memoryTypeBits & VK_MEMORY_PROPERTY_HOST_CACHED_BIT);

std::vector<NvSciBufAttrKeyValuePair> pairArray = {
{ NvSciBufImageAttrKey_Layout, (void*) &layout, sizeof(layout) },
{ NvSciBufImageAttrKey_PlaneCount, (void*) &planeCount, sizeof(planeCount) },
{ NvSciBufImageAttrKey_PlaneColorFormat, (void*) &colorFormat, sizeof(colorFormat) },
}
```

```

{ NvSciBufImageAttrKey_PlaneHeight, (void*) &height, sizeof(height) },
{ NvSciBufImageAttrKey_PlaneWidth, (void*) &width, sizeof(width) },
{ NvSciBufImageAttrKey_PlaneScanType, (void*) &planescantype, sizeof(planescantype) },
{ NvSciBufGeneralAttrKey_Types, (void*) &bufType, sizeof(bufType) },
{ NvSciBufGeneralAttrKey_NeedCpuAccess, (void*) &needCpuAccess, sizeof(needCpuAccess) },
{ NvSciBufGeneralAttrKey_EnableCpuCache, (void*) &needCpuCached,
  sizeof(needCpuCached) },
{ NvSciBufGeneralAttrKey_RequiredPerm, (void*) &perm, sizeof(perm) },
};

NvSciBufAttrList attrList, reconciledList, conflictList;
NvSciBufAttrListCreate(scibufModule, &attrList);

// Application fills the public key-value pairs
NvSciBufAttrListSetAttrs(*attrList, pairArray.data(), pairArray.size());
vkGetPhysicalDeviceSciBufAttributesNV(physDev, *attrList);

// Allocate NvSciBufObj
NvSciBufAttrListReconcile(&attrList, 1, &reconciledList, &conflictList);
NvSciBufObj sciBufObjVkImage {nullptr};
NvSciBufObjAlloc(reconciledList, &sciBufObjVkImage);

// Create VkDeviceMemory
VkImportMemorySciBufInfoNV importSciBufInfo {
    .sType = VK_STRUCTURE_TYPE_IMPORT_MEMORY_SCI_BUF_INFO_NV;
    .pNext = nullptr;
    .handleType = VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV;
    .handle = sciBufObjVkImage;
};

// GetMemoryType is a helper function to convert memoryTypeBits to memoryType index,
// skip the detail of implementation here.
VkMemoryAllocateInfo memAllocInfo = {
    .sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
    .pNext = importSciBufInfo;
    .memoryTypeIndex = needCpuAccess ?
        GetMemoryType(memReqs.memoryTypeBits, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT):
        GetMemoryType(memReqs.memoryTypeBits, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
    .allocationSize = 0; // ignore the allocate size when importing NvSciBufObj
};

VkDeviceMemory devMemory;
vkAllocateMemory(dev, &memAllocInfo, NULL, &devMemory);
vkBindImageMemory(dev, sciBufObjVkImage, devMemory, 0);

// De-init
vkDestroyImage(dev, vkimage, 0);
NvSciBufAttrListFree(attrList);
NvSciBufAttrListFree(reconciledList);
NvSciBufAttrListFree(conflictList);
NvSciBufObjFree(sciBufObjVkImage);

```

6.6.3.9 Late Attach

`NvSciBuf` provides a mechanism for an application to allocate an `NvSciBufObj` that is shared with other peer `NvSciIpcEndpoints` after allocation of `NvSciBufObj`, without requiring each peer `NvSciIpcEndpoint` to send an unreconciled `NvSciBufAttrList` during initial reconciliation of unreconciled `NvSciBufAttrList` and allocation of `NvSciBufObj`.

To ensure that the allocated `NvSciBufObj` satisfies the memory requirements of the late peer `NvSciIpcEndpoint`, Application must create a proxy unreconciled `NvSciBufAttrList` representing the memory requirements of the late peer `NvSciIpcEndpoint`. This proxy unreconciled `NvSciBufAttrList` must be involved during initial reconciliation of the unreconciled `NvSciBufAttrList`. `NvSciBuf` provides attribute keys such as `NvSciBufAttrKey_PeerLocationInfo` and `NvSciBufGeneralAttrKey_PeerHwEngineArray` that should be specified in the proxy unreconciled `NvSciBufAttrList` during initial reconciliation.

`NvSciBuf` also allows remote late peer `NvSciIpcEndpoint` to gain access to already allocated `NvSciBufObj`, provided the allocated `NvSciBufObj` satisfies the constraints of the unreconciled `NvSciBufAttrList`, received from remote late peer `NvSciIpcEndpoint`, through the `NvSciBufObjAttachPeer()` API.

If the `NvSciBufGeneralAttrKey_PeerHwEngineArray` attribute is not specified in the proxy unreconciled `NvSciBufAttrList`, then `NvSciBuf` allocates a buffer of greater than the requested size to compensate for hardware engines that gain access to memory after allocation of `NvSciBufObj`. `NvSciBuf` also chooses the appropriate carveout memory location, which is accessible to hardware engines that gain access to memory after allocation of `NvSciBufObj`.



Note:

- > Detailed descriptions of the attribute keys and the APIs are in the *NVIDIA DRIVE® OS API Reference*.
- > Application needs to refer to User Mode Driver (UMD) documentation for details of how to specify the `NvSciBufGeneralAttrKey_PeerHwEngineArray` key.
- > The `NvSciBufGeneralAttrKey_PeerHwEngineArray` attribute is experimental in NVIDIA DRIVE OS 6.0.7 and should not be used in a production environment.

6.6.4 Streaming

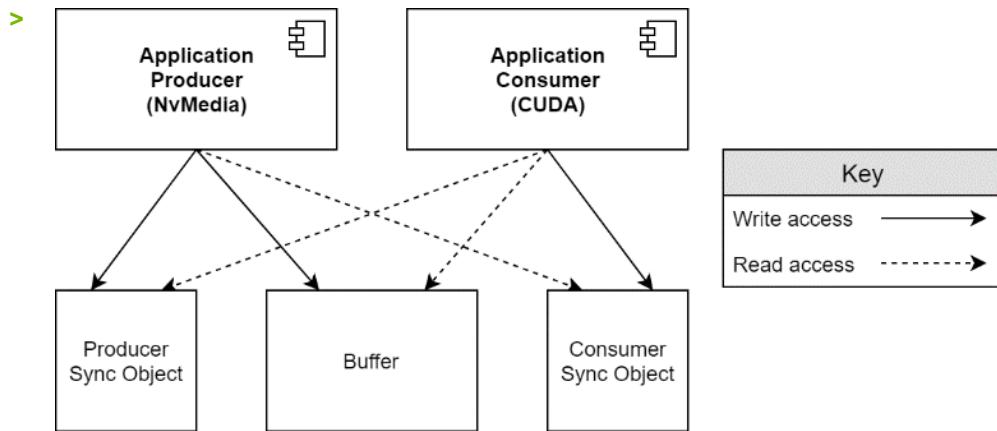
Combining the buffer and synchronization functions from the previous chapters allows you to develop applications that stream sequences of data from one rendering component to another, building an efficient processing pipeline. For developers who wish to have complete control over the process, no additional functionality is required.

However, use cases for streaming can become quite complex, making the details difficult to manage. This is particularly true when portions of the pipeline are provided by independent developers who must coordinate the stream management. NVIDIA

therefore provides an additional NvSciStream library layered on NvSciBuf and NvSciSync with utilities for constructing streaming application suites.

6.6.4.1 A Simple Stream

This section illustrates how streaming works with an example application that directly uses NvSciBuf and NvSciSync to take images from a camera controlled by NvMedia and sends the images to CUDA for processing. This application is so simple that there is no need to involve the added NvSciStream layer. It uses a single buffer that NvMedia can write to and CUDA can read from, and a pair of sync objects. One sync object is for NvMedia to write and CUDA to read, and the other is for CUDA to write and NvMedia to read.



6.6.4.1.1 Setup

The initial setup for the application is shown in the following topic. Much of this should already be familiar from the earlier chapters on buffers and synchronization objects. (For brevity, we assume everything succeeds and omit error checking.)

6.6.4.1.1.1 Simple Stream Setup

```

/* Initialize NvMedia (not all steps are shown) */
NvMedia2D* nvmedia2D;
NvMedia2DCreate(nvmedia2D, NULL);

/* Initialize CUDA (not all steps are shown) */
CUdevice cudaDevice;
cuDeviceGet(&cudaDevice, IGPU);
CUcontext cudaContext;
cuCtxCreate(&cudaContext, CU_CTX_MAP_HOST, dev);
cuCtxPushCurrent(&cudaContext);
CUstream cudaStream;
cuStreamCreate(&cudaStream, CU_STREAM_DEFAULT);

/* Initialize NvSci buffer and sync modules */
NvSciBufModule bufModule;
NvSciBufModuleOpen(&bufModule);
NvSciSyncModule syncModule;
NvSciSyncModuleOpen(&syncModule);
  
```

```

/* Obtain NvMedia buffer requirements */
NvSciBufAttrList nvmediaBufAttrs;NvSciBufAttrListCreate(bufModule, &nvmediaBufAttrs);
NvMedia2DFillNvSciBufAttrList(nvmedia2D, nvmediaBufAttrs);

/*
 * Set more buffer attributes using NvSciBufAttrListSetAttrs.
 * Detail skipped.
 */

/* Obtain NvMedia sync requirements */
NvSciSyncAttrList nvmediaWriteSyncAttrs,
nvmediaReadSyncAttrs;
NvSciSyncAttrListCreate(syncModule,
&nvmediaWriteSyncAttrs);
NvMedia2DFillNvSciSyncAttrList(nvmediaWriteSyncAttrs,
NVMEDIA_SIGNALER)
NvSciSyncAttrListCreate(syncModule,
&nvmediaReadSyncAttrs);
NvMedia2DFillNvSciSyncAttrList
(nvmediaReadSyncAttrs, NVMEDIA_WAITER)

/* Obtain CUDA buffer requirements */
NvSciBufAttrList cudaBufAttrs;
NvSciBufAttrListCreate(bufModule, &cudaBufAttrs);
<Fill in with CUDA raw buffer attributes>

/* Obtain CUDA sync requirements */
NvSciSyncAttrList cudaWriteSyncAttrs,
cudaReadSyncAttrs;
NvSciSyncAttrListCreate(syncModule,
&cudaWriteSyncAttrs);
cuDeviceGetNvSciSyncAttributes(cudaWriteSyncAttrs,
cudaDevice, CUDA_NVSCISYNC_ATTR_SIGNAL);
NvSciSyncAttrListCreate(syncModule,
&cudaReadSyncAttrs);
cuDeviceGetNvSciSyncAttributes(cudaReadSyncAttrs,
cudaDevice, CUDA_NVSCISYNC_ATTR_WAIT);

/* Combine buffer requirements and allocate buffer */
NvSciBufAttrList allBufAttrs[2], conflictBufAttrs;
NvSciBufAttrList combinedBufAttrs;
allBufAttrs[0] = nvmediaBufAttrs;
allBufAttrs[1] = cudaBufAttrs;
NvSciBufAttrListReconcile(allBufAttrs, 2,
&combinedBufAttrs, &conflictBufAttrs);
NvSciBufObj buffer;
NvSciBufObjAlloc(combinedBufAttrs, &buffer);

/* Combine sync requirements and allocate
nvmedia to cuda sync object */
NvSciSyncAttrList allSyncAttrs[2], conflictSyncAttrs;
allSyncAttrs[0] = nvmediaWriteSyncAttrs;

```

```

allSyncAttrs[1] = cudaReadSyncAttrs;
NvSciSyncAttrList nvmediaToCudaSyncAttrs;
NvSciSyncAttrListReconcile(allSyncAttrs, 2,
&nvmediaToCudaSyncAttrs, &conflictSyncAttrs);
NvSciSyncObj nvmediaToCudaSync;
NvSciSyncObjAlloc(nvmediaToCudaSyncAttrs,
&nvmediaToCudaSync);

/* Combine sync requirements and allocate cuda
to nvmedia sync object */
allSyncAttrs[0] = cudaWriteSyncAttrs;
allSyncAttrs[1] = nvmediaReadSyncAttrs;
NvSciSyncAttrList cudaToNvmediaSyncAttrs;
NvSciSyncAttrListReconcile(allSyncAttrs, 2,
&cudaToNvmediaSyncAttrs, &conflictSyncAttrs);
NvSciSyncObj cudaToNvmediaSync;
NvSciSyncObjAlloc(cudaToNvmediaSyncAttrs, &cudaToNvmediaSync);

/* Map objects into NvMedia */
NvMedia2DRegisterNvSciBufObj(nvmedia2D, buffer);
NvMedia2DRegisterNvSciSyncObj(nvmedia2D, NVMEDIA_EOFSYNCOBJ, nvmediaToCudaSync);
NvMedia2DRegisterNvSciSyncObj(nvmedia2D, NVMEDIA_PRESYNCOBJ, cudaToNvmediaSync);

/* Map objects into CUDA */
cudaExternalMemoryHandleDesc
cudaMemHandleDesc;
memset(&cudaMemHandleDesc, 0, sizeof
(cudaMemHandleDesc));
cudaMemHandleDesc.type =
cudaExternalMemoryHandleTypeNvSciBuf;
cudaMemHandleDesc.handle.nvSciBufObject =
buffer;
cudaMemHandleDesc.size = <allocated size>;
cudaImportExternalMemory(&cudaBuffer,
&cudaMemHandleDesc);
CUDA_EXTERNAL_SEMAPHORE_HANDLE_DESC
cudaSemDec;
CUExternalSemaphore nvmediaToCudaSem,
cudaToNvmediaSem;
cudaSemDesc.type =
CU_EXTERNAL_SEMAPHORE_HANDLE_TYPE_NVSCISYNC;
cudaSemDesc.handle.nvSciSyncObj =
(void*)nvmediaToCudaSync;
cuImportExternalSemaphore(&nvmediaToCudaSem,
&cudaSemDesc);
cudaSemDesc.type =
CU_EXTERNAL_SEMAPHORE_HANDLE_TYPE_NVSCISYNC;
cudaSemDesc.handle.nvSciSyncObj =
(void*)cudaToNvmediaSync;
cuImportExternalSemaphore(&cudaToNvmediaSem,
&cudaSemDesc);

```

First, the buffer and sync object requirements are queried from NvMedia, the producer of the stream, and from CUDA, the consumer. These requirements are combined and used to allocate the objects, which are then mapped into NvMedia and CUDA so that they can be used for processing.

Two sync objects are required instead of one because synchronization is required in both directions. It is important that the CUDA consumer does not begin reading from the buffer until the NvMedia producer is done writing to it. It is equally as important that the NvMedia producer does not begin writing a new image to the buffer until the CUDA consumer is done reading the previous image. Otherwise, it overwrites data that is still in use.

6.6.4.1.2 Streaming

Once initialized, the streaming loop looks like this:

```

/* Initialize empty fences for each direction*/
NvSciSyncFence nvmediaToCudaFence = NV_SCI_SYNC_FENCE_INITIALIZER;
NvSciSyncFence cudaToNvmediaFence = NV_SCI_SYNC_FENCE_INITIALIZER;

/* Main rendering loop */
while (!done) {
    NvMedia2DComposeResult result;
    NvMedia2DComposeParameters params;
    NvMedia2DGetComposeParameters(nvmedia2D, &params);

    /* Generate a fence when rendering finishes */
    NvMedia2DSetNvSciSyncObjforEOF(nvmedia2D, params, nvmediaToCudaSync);

    /* Instruct NvMedia pipeline to wait for the fence from CUDA */
    NvMedia2DInsertPreNvSciSyncFence(nvmedia2D, params, cudaToNvmediaFence)
        /* Generate NvMedia image */
        NvMedia2DSomeRenderingOperation( ...);
        NvMedia2DCompose(nvmedia2D, params, &result);

    NvMedia2DGetEOFNvSciSyncFence(nvmedia2D, result, &nvmediaToCudaFence);

    /* Instruct CUDA pipeline to wait for fence from NvMedia */
    CUDA_EXTERNAL_SEMAPHORE_WAIT_PARAMS cudaWaitParams;
    cudaWaitParams.params.nvSciSync.fence = (void*)&nvmediaToCudaFence;
    cudaWaitParams.flags = 0;
    cudaWaitExternalSemaphoresAsync(&nvmediaToCudaSem, &cudaWaitParams, 1, cudaStream);

    /* Process the frame in CUDA */
    cudaSomeProcessingOperation(..., cudaBuffer, ...);

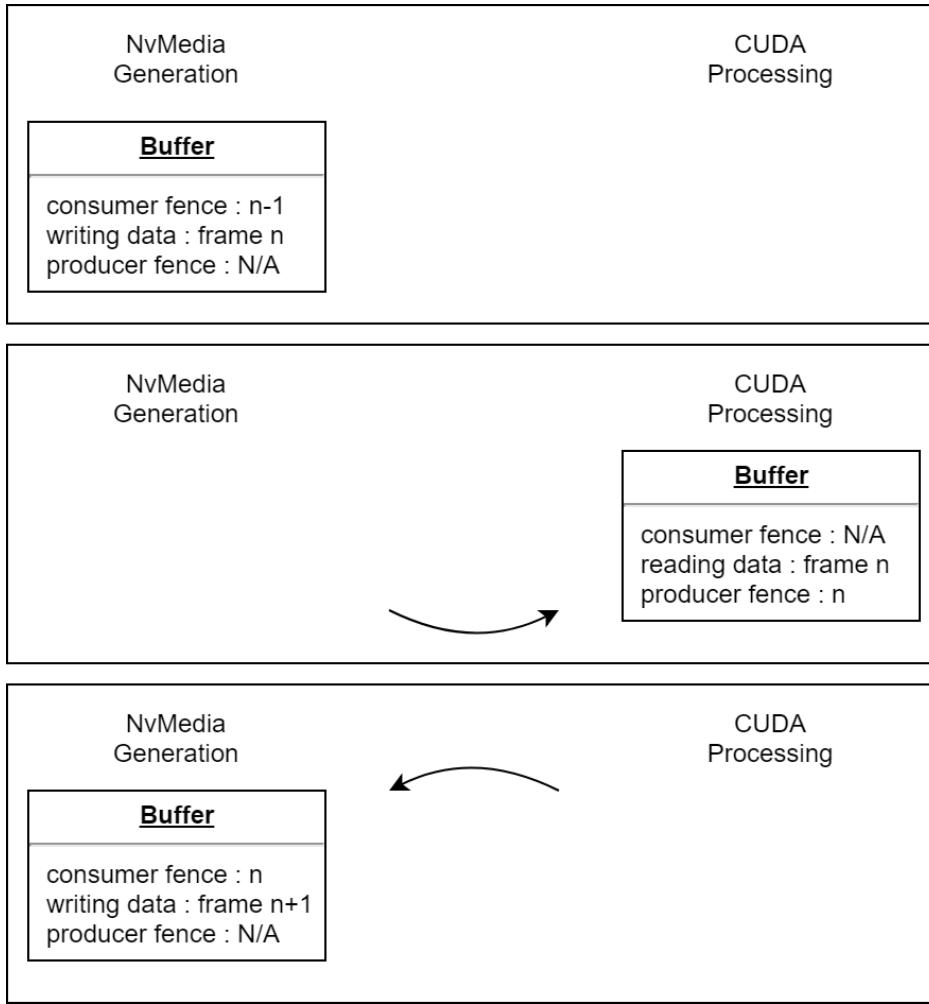
    /* Generate a fence when processing finishes */
    CUDA_EXTERNAL_SEMAPHORE_SIGNAL_PARAMS cudaSignalParams;
    cudaSignalParams.params.nvSciSync.fence = (void*)&cudaToNvmediaFence;
    cudaSignalParams.flags = 0;
    cudaSignalExternalSemaphoresAsync(&cudaToNvmediaSem, &cudaSignalParams, 1,
    cudaStream);

```

}

For each frame of data, the application instructs NvMedia to write the image to the buffer, and then issues a fence that indicates when writing finishes. CUDA is instructed to wait for that fence before it proceeds with any subsequent operations, then the commands to process the frame are issued to CUDA. Lastly, CUDA is told to generate a fence of its own, which indicates when all its operations finish. This fence is fed back to NvMedia, which waits for it before starting to write the next image to the buffer. "Ownership" of the buffer cycles back and forth between producer and consumer.

>



6.6.4.1.2.1 NvStreams Rawstream Sample Application

The NVIDIA SDK provides a sample application to demonstrate how to use the NvSciStream API to build simple and complex streams, see the NvSciStream Sample Application chapter. This Rawstream Sample Application doesn't use NvSciStream. Instead, it builds a simple inter-process stream from CUDA to CUDA using NvSciBuf, NvSciSync and NvScilpc APIs directly.

Prerequisites

NvScilpc

The sample applications stream packets between a producer process and a consumer process via an inter-process communication (NvScilpc) channel.

The NvScilpc channels are configured via the device tree (DT) on QNX and via a plain text file, /etc/nvscilpc.cfg, on Linux. For more information on NvScilpc configuration data, see [NvScilpc Configuration Data](#). The recommended NvScilpc channels for this sample application is as follows:

```
INTER_PROCESS nvscisync_a_0 nvscisync_a_1 16 24576
```

CUDA

This sample application uses the CUDA toolkit. Ensure the CUDA toolkit is installed. For more information, see [Installing CUDA Debian Packages](#)

Building the Rawstream Sample Application

The Rawstream sample includes source code and a Makefile.

On the host system, navigate to the sample application directory:

```
cd <top>/drive-linux/samples/nvsci/rawstream/
```

Build the sample application:

```
make clean
```

```
make
```

Running the Rawstream Sample Application

Copy the sample application to the target filesystem:

```
cp <top>/drive-linux/samples/nvsci/rawstream/rawstream <top>/drive-linux/
targetfs/home/nvidia/
```

Run the sample application:

```
./rawstream -p &
```

```
./rawstream -c
```



Note:

The rawstream application must be run as root user (with sudo).

If the rawstream application fails to open the IPC channel, cleaning up NvScilpc resources may help.

```
sudo rm -rf /dev/mqueue/*
```

```
sudo rm -rf /dev/shm/*
```

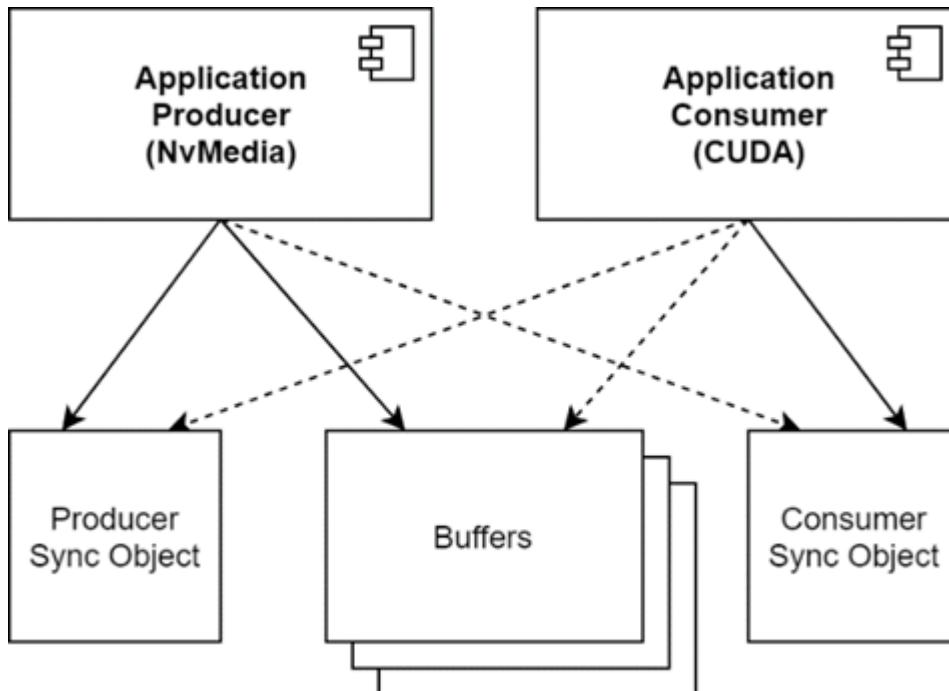
6.6.4.2 More Complex Streams

All the basic steps in the previous example to set up buffers and sync objects, and to generate and wait for fences must be performed for all streaming applications, regardless of whether they use NvSciStream. For a simple situation like that, there is no need for anything more.

What NvSciStream provides is the ability to manage more complex cases, such as cycling between multiple buffers, streaming to multiple consumers at once, and streaming between applications. It also provides a uniform set of interfaces for setting up streams for which independent developers can design modular producers and consumers without needing to directly coordinate and then plug their products together. This section describes some of those use cases Adding NvSciStreams for these use cases relieves the developer of many burdensome details.

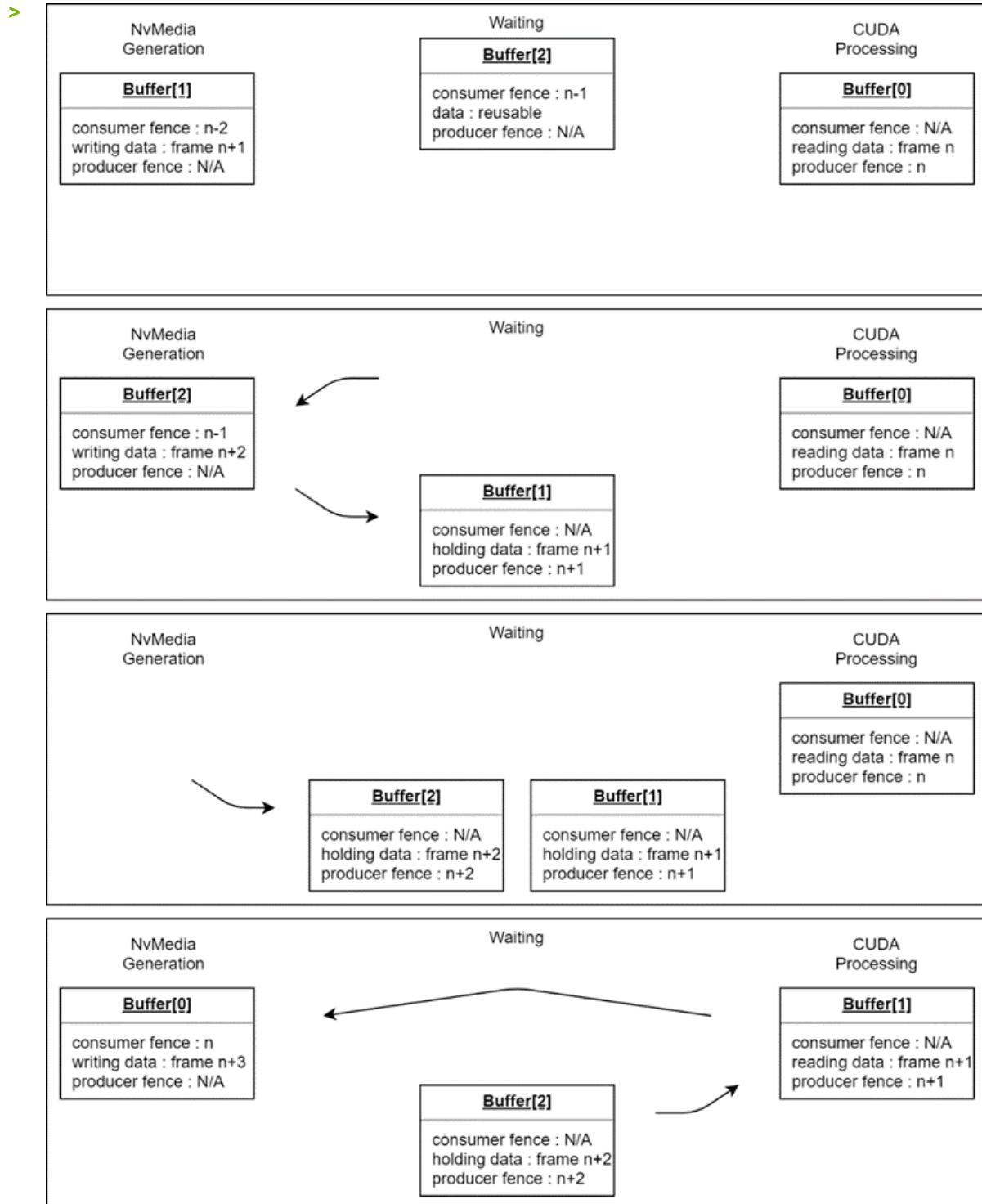
6.6.4.2.1 Multiple Buffers

NvMedia and CUDA use different portions of the NVIDIA hardware. In the example above, the compute hardware is idle during 2D processing, and the 2D hardware is idle during compute processing. The pipeline can be made more efficient by allocating one (1) or more additional buffers and cycling between them, creating a queue of frames. This way, CUDA can process one image while NvMedia prepares the next one. No additional sync objects are needed, but it is necessary to generate a fence for each frame, and therefore you must keep track of which fence is associated with the current contents of each buffer. Once a stream has multiple buffers available, there are several different possible modes of operation to consider.



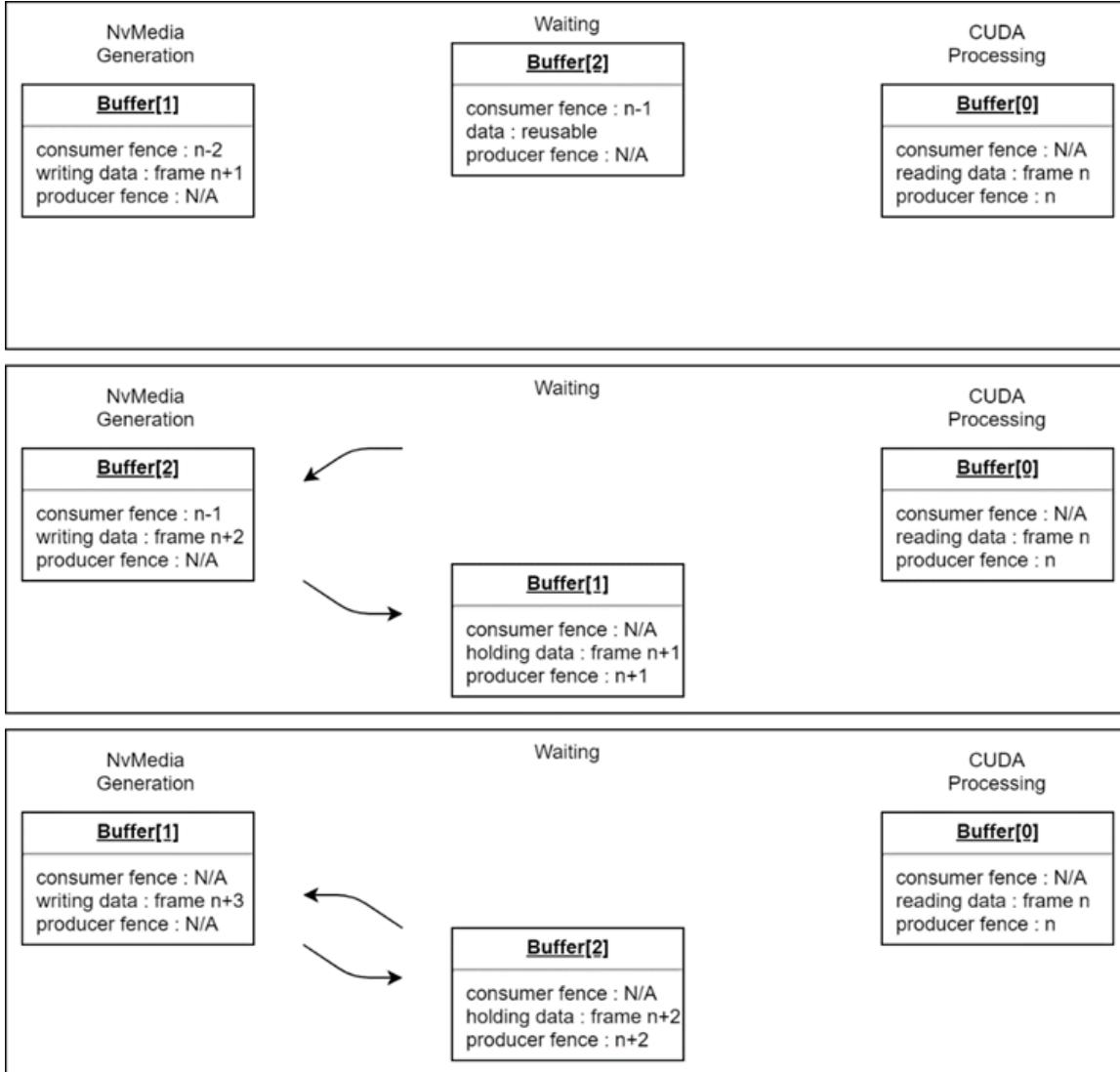
6.6.4.2.2 FIFO Mode

If the use case requires that all data in the sequence be processed, the stream application operates in FIFO mode. When the producer fills a buffer with new data, it must wait for the consumer to process it. If the consumer requires more time than the producer, the producer is slowed to the consumer's speed to wait for buffers to become available for reuse.



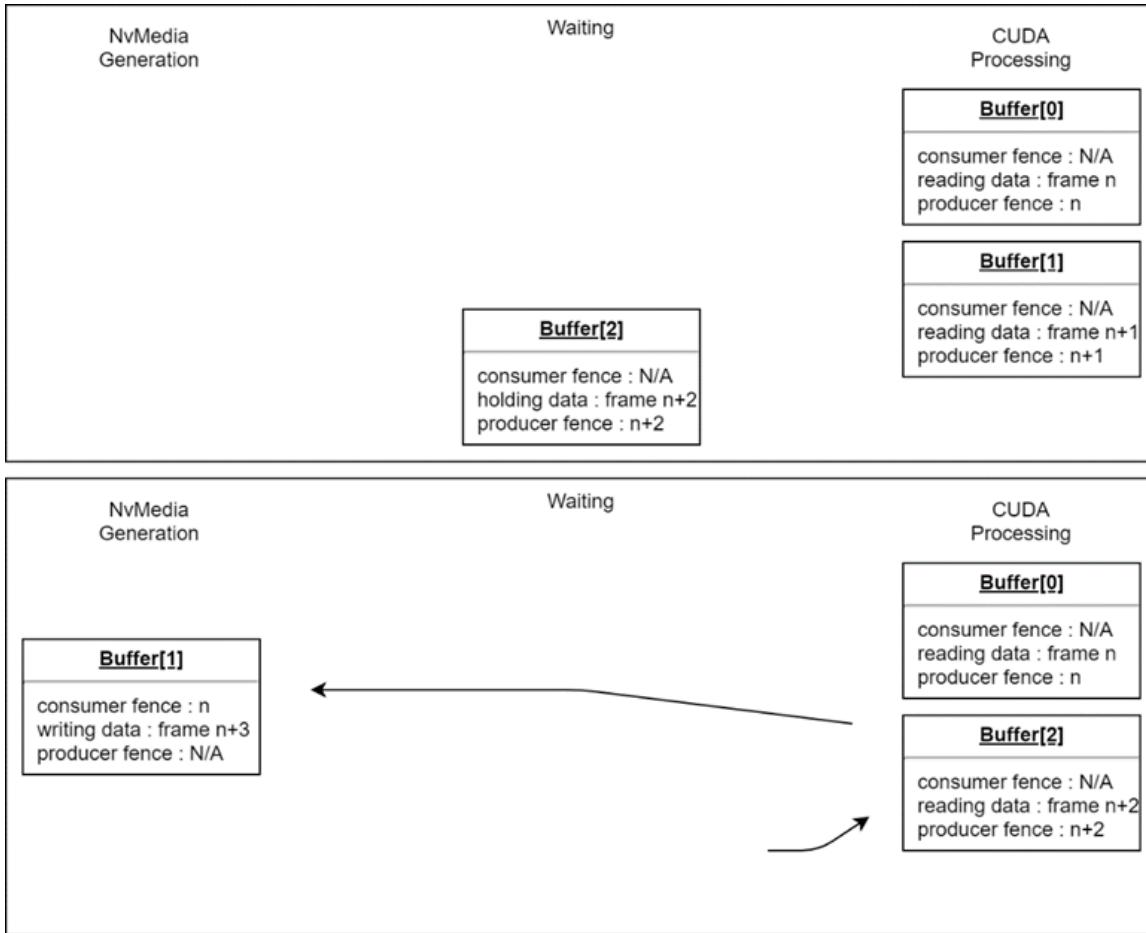
6.6.4.2.3 Mailbox Mode

In other use cases, it is more important that the consumer always have the most recent input available. In this case, the stream application operates in mailbox mode. If the consumer has not started processing a previous frame when a new one becomes available, it is skipped, and its buffer immediately returned to the producer for reuse. This means that the order in which the producer and consumer cycle through buffers is subject to change at any time, and the streaming system must manage this.



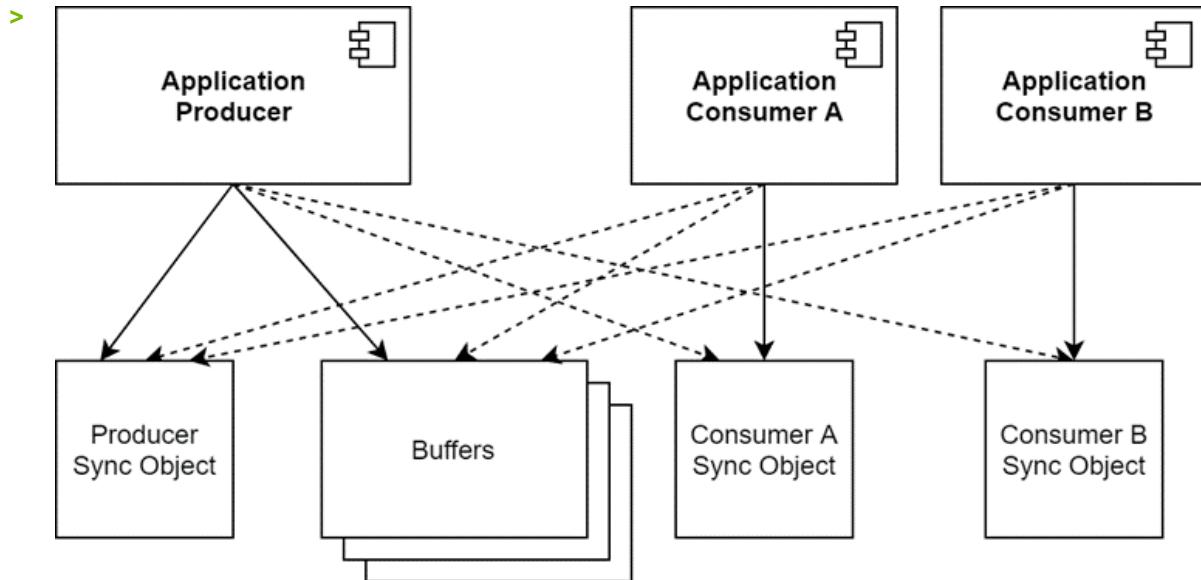
6.6.4.2.4 Multiple Acquired Frames

Some processing algorithms must compare data from multiple frames in a sequence. In this case, the consumer may hold multiple buffers at once. These buffers may be released for reuse in an order other than they arrive. The streaming system must manage this.

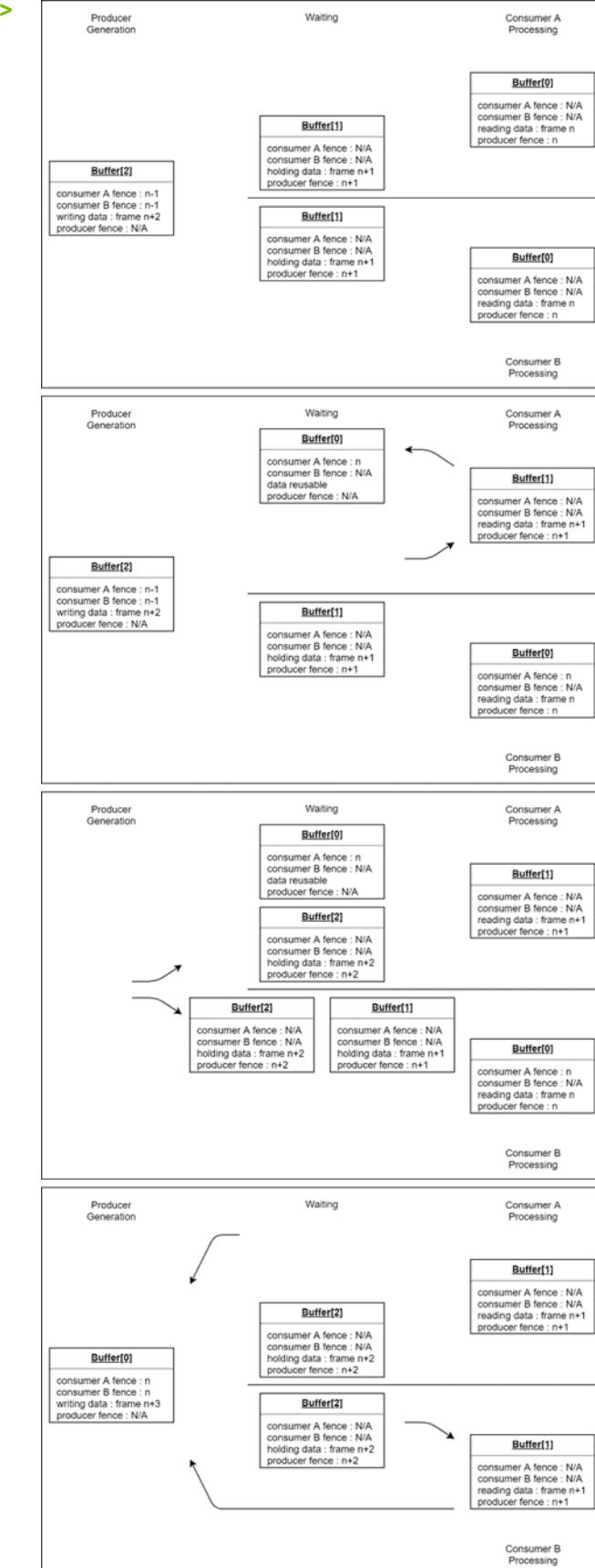


6.6.4.2.5 Multiple Consumers

In some cases, the output of a single producer must be sent to multiple consumers. This means that during initialization, the requirements from all of the consumers must be gathered, and sync objects from all of the consumers must be mapped into the producer.



During streaming, the system must track each buffer's state with regards to all the consumers and wait for fences from all of them to complete before the producer can reuse it. This can be further complicated if one consumer requires FIFO behavior and another requires mailbox. They might return buffers in different orders.



6.6.4.2.6 Limit Acquired Frames

In the multiple consumers case, there is a security and robustness concern that if one of the consumer applications either maliciously or through some bug stops returning packets for reuse, it will bring the producer and all other consumers to a halt because they will run out of space for the new data payloads. One way to mitigate this risk is to place an optional cap on the number of packets that will be allowed to be sent downstream to the unreliable application.

During streaming, if a new packet is available but an untrusted application is already at its cap, the packet will immediately be returned upstream without the consumer ever seeing it.

6.6.4.2.7 Cross-Application

Having producers and consumers in separate applications requires inter-process communication during every step of the process. At setup, all the requirements must be transmitted between the endpoints, and then the allocated objects must be shared between the processes. During streaming, every time the producer generates a frame, a message letting the consumer know it is ready, which includes the fence, must be sent to the other side. When the consumer is done reading from the frame, a similar message must be sent back in the other direction. Each application must be prepared to read and process these messages so streaming can occur in a timely fashion.

6.6.4.2.8 Chip to Chip (C2C)

Producers and consumers running on separate applications on different SoCs requires inter-SoC communication. The data transmission is across memory boundaries. The streaming is no longer a copyless operation. In addition to the requirements for cross-application use cases, the buffer data of each frame is copied from the producer system to the consumer system. The receiving end (not necessarily the consumer process) of the inter-SoC communication requires allocation of buffers as the destination of C2C data copy.

6.6.4.2.9 Forced Synchronization of Payloads

Certain sections in a stream may require the data passing through be synchronous, meaning when the payloads arrive in those sections all producers or consumers have done writing to or reading from the buffer data in the payloads. Before reaching those sections in the stream, payloads that are accompanied with fences have to be blocked until the fences expire.

6.6.4.2.10 Late-Attach Consumers

In a use case with multiple consumers, if the consumer is on the same SoC as the producer, it can connect to the stream quickly. However, some consumers on another SoC may be launched late and connected to the stream late. The application may want to start transmitting payloads to the early-connected consumers and only attach the late consumers once they are ready.

6.6.4.2.11 Re-Attach Consumers

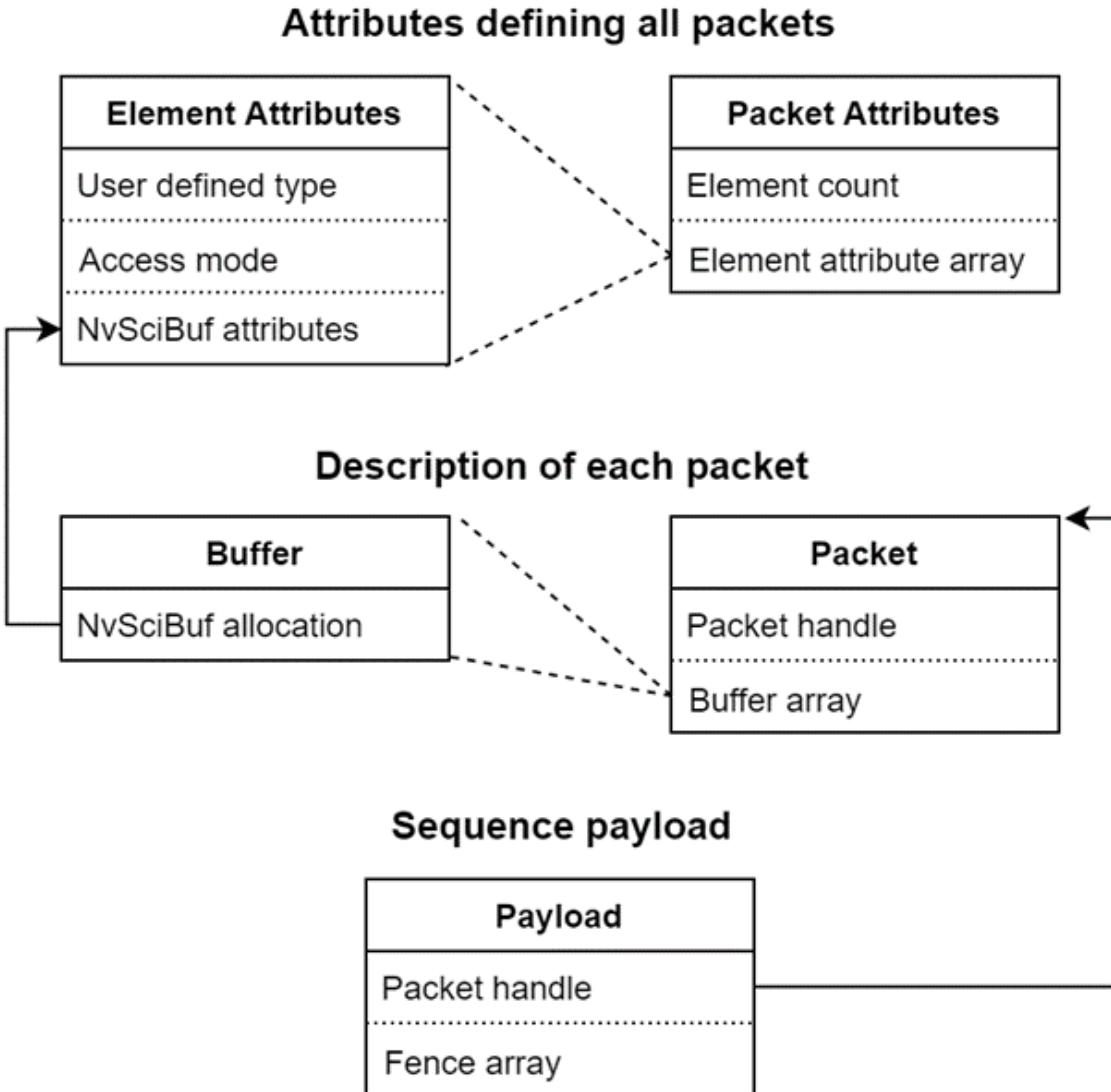
In some use cases, one consumer process may crash or hang unexpectedly. The application may want to restart this consumer process and re-attach it to the same stream without impacting other consumers.

6.6.4.3 Data Packets

In the previous examples, only 2D buffers containing single frames of image data are considered. For many use cases, that is all that is required. However, NvSciStream supports more arbitrary data. In the sections below, "buffers" are generalized to "packets", and "frames" to "payloads".

A packet is a set of indexed buffers that contains images, tensors, metadata, or other information. These buffers are referred to as the "elements" of a packet. Each stream may use one or more packets. For a given stream, all packets have the same number of elements, and the i th element of all packets is allocated with the same NvSciBuf attributes. As a result, all packets have uniform memory requirements and signatures. Producers and consumers send and receive entire packets at a time, along with associated fences to indicate when the payloads they contain are ready.

In addition to the NvSciBuf attributes used to allocate them, the elements of each packet also have a type and NvSciSync attributes associated with them, which help applications coordinate how the elements are used. These are described in the next sections.



6.6.4.3.1 Element Type

In a complex modular application suite, a given producer may know how to produce many different types of data, and the various consumers may only require a subset of them. NvSciStream provides mechanisms to help applications negotiate the data that is required for a given stream. The top-level system integrator must define a set of integer values to associate with each type of data that the application suite supports. Any non-zero value may be used for these types, and developers are advised to plan for future types and backwards compatibility.

As an example, the following table is a subset of the types that are available in an automotive system. These may not all be supplied by a single producer. For instance, one producer might be responsible for front-facing sensors, and another for rear-facing sensors. Different consumers make use of different sets of sensors to generate their output. The types listed in the table below all correspond to raw inputs, but a pipe-lined application may have a series of producers and consumers with additional types to

represent intermediate processed data. The following table describes the sample data packet types:

Data Type	Assigned Enum
Front left camera	0x1101
Front-center camera	0x1102
Front-right camera	0x1103
Rear-left camera	0x1201
Rear-center camera	0x1202
Rear-right camera	0x1203
Front radar	0x2110
Front lidar	0x2120
Rear radar	0x2210
Rear lidar	0x2220
GPS	0x3010
IMU	0x3020

At initialization, a producer declares all the types of data it can provide. Consumers indicate the types of data they are interested in. The part of the application responsible for allocating the buffers takes this information and collates it to determine the packet layout and passes the packet attributes back to the producer and consumers. The producer checks this layout to see what elements it needs to provide. The consumers check it to determine where in the packets to find the elements they care about. Any element that a given consumer does not require can be ignored.

In simple applications that only deal with a single type of data such as images, producers and consumers can simply specify a value of 1 for the type. These type exchange mechanisms are only intended to aid integration of larger suites.

6.6.4.3.2 Element Mode

Payload data generated and processed by NVIDIA hardware is usually written and read asynchronously and requires waiting for a fence before it can be accessed. But in some use cases, auxiliary data may be generated synchronously by the CPU and must be read

before the commands to process the rest of the data are issued. An example is a camera producer that generates images asynchronously, but also includes a synchronously generated metadata field that contains the camera's focal length, exposure time, and other settings. A consumer synchronously reads in the metadata first and uses the values it contains when issuing the commands to asynchronously process the image.

The endpoint that requires an element to be synchronous must provide the `NULL` `waiter` `sync` attribute to the opposing endpoint for that element.

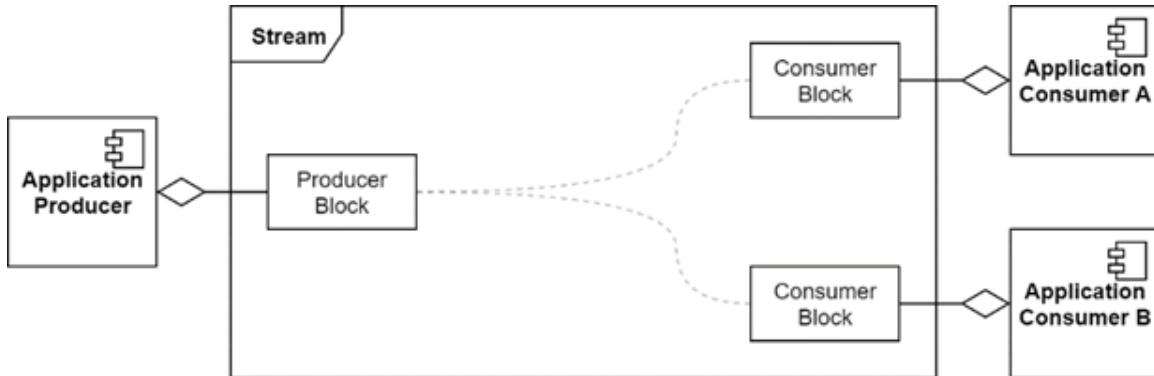
6.6.4.4 Building Block Model

NvSciStream is designed to make implementing complex stream use cases easier by providing a uniform set of interfaces for exchanging requirements and sending and receiving frames. At the top level, an application suite determines the overall structure of the stream. At the lower application component level, each producer and consumer performs its own function, without needing to know the details of the full stream. They can be developed independently in a modular fashion and plugged together as needed.

In order to support this wide variety of use cases, NvSciStream itself takes a modular approach. It defines a set of building blocks, each with a specific purpose, which applications create and connect together to suit their needs. This section describes the available building blocks and their behavior.

6.6.4.4.1 Endpoints

Each stream begins with a producer block and ends with one or more consumer blocks. These are referred to as the "endpoints" of the stream, with the producer being the "upstream" end and the consumers being the "downstream" end. For each stream endpoint, there is an application component responsible for interacting with it. The endpoints and their corresponding application components may all reside in a single process or be distributed across multiple processes, partitions, or systems.



6.6.4.4.2 Producer

A producer block provides the following interactions with the application producer component:

- > During setup phase:
 - Synchronization:

- Accepts producer synchronization requirements.
- Reports consumer synchronization requirements.
- Accepts producer allocated sync objects.
- Reports consumer allocated sync objects.
- Buffers
 - Accepts list of element attributes producer can generate.
 - Reports consolidated packet attribute list.
 - Reports all allocated packets.
- > During streaming phase:
 - Signals availability of packets for reuse.
 - Retrieves next packet to reuse with fence to wait for before writing.
 - Accepts filled packet with fence to wait for before consumer should read.
- > In non-safety builds, during streaming phase:
 - Accepts limited changes to requested element attributes (e.g., to modify the image size).
 - Reports updated packet attributes.
 - Reports removal of buffers allocated for old attributes and addition of new ones.



Note:

Reallocation of buffers in non-safety builds are not supported in this release.

6.6.4.4.3 Consumer

A consumer block provides the following interactions with its corresponding application consumer component:

- > During setup phase:
 - Synchronization:
 - Accepts consumer synchronization requirements.
 - Reports producer synchronization requirements.
 - Accepts consumer allocated sync objects.
 - Reports producer allocated sync objects.
 - Buffers:
 - Accepts list of element attributes consumer requires.
 - Reports consolidated packet attribute list.
 - Reports all allocated packets.
- > During streaming phase:
 - Signals availability of packets for reading.
 - Retrieves next payload with fence to wait for before reading.
 - Accepts packet, which can be reused with fence to wait for before the producer writes new data.
- > In non-safety builds, during streaming phase:

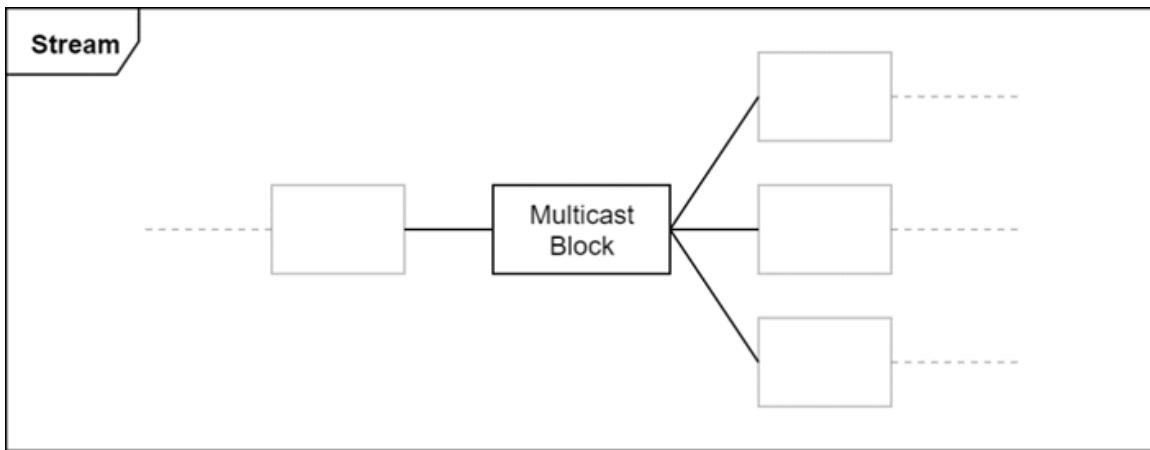
- Reports updated packet attributes.
- Reports removal of buffers allocated for old attributes and addition of new ones.

**Note:**

Reallocation of buffers in non-safety builds are not supported in this release.

6.6.4.4.4 Multicast

When a stream has more than one consumer, a multicast block is used to connect the separate pipelines. This block distributes the producer's resources and actions to the consumers, who are unaware that each other exist. It combines the consumers' resources and actions, making them appear to the producer as a single virtual consumer. The block does not directly provide any mechanisms to safeguard any of its consumers against faulty or malicious behavior in its other consumers. Additional blocks can be used to isolate individual consumer pipelines from others when there are safety or security concerns.



Once created and connected, no direct interaction by the application with a multicast block is required. It automatically performs the following operations as events arrive from up and downstream:

- > During setup phase:
 - Synchronization:
 - Passes producer synchronization requirements to all consumers.
 - Combines consumer synchronization requirements and passes to producer.
 - Passes producer allocated sync objects to all consumers.
 - Passes sync objects allocated by all consumers to producer.
 - Buffers:
 - Combines lists of element attributes from all consumers into a single list and passes upstream.
 - Passes consolidated packet attribute list to all consumers.
 - Passes allocated packets to all consumers.
- > During streaming phase:

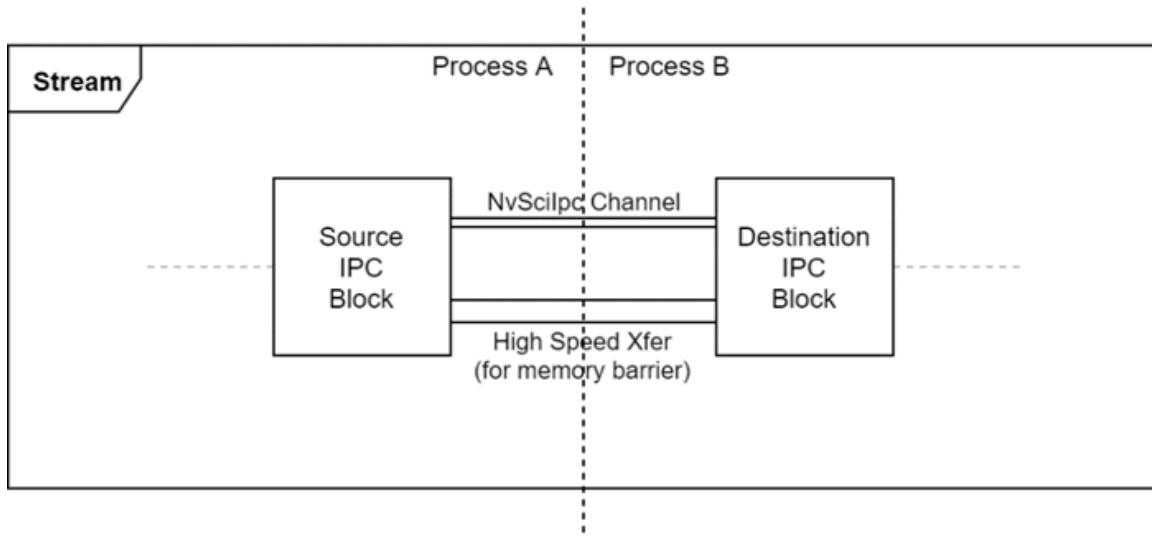
- Passes available payloads to all consumers.
 - Tracks packets returned for reuse by the consumers and returns them to the producer when all consumers are done with them. All fences are combined into one list.
- > In non-safety builds, during streaming phase:
- Distributes changes in packet attributes and buffers to all consumers.

**Note:**

Reallocation of buffers in non-safety builds are not supported in this release.

6.6.4.4.5 IPC

When the endpoints of a stream reside in separate processes, IPC blocks are used to bridge the gaps. They are created in pairs, with a source IPC block in the upstream process and a destination IPC block in the downstream process. The communication must first be established by the application using NvScilpc, and then the channel endpoints are passed to the two IPC blocks. They take ownership of the channel and use it to coordinate the exchange of requirements and resources and signal the availability of packets.



There are two types of IPC pairs available, depending on whether the upstream and downstream portions share memory.

6.6.4.4.5.1 Memory Sharing IPC

When the two halves of the stream access the same physical memory, memory sharing IPC blocks can be used. These coordinate the sharing of resources between the two ends but do not need to access the payload data. They perform the following actions when events arrive from up and downstream:

- > During setup phase:
- Synchronization:

- Exports synchronization requirements from the producer and consumer(s) and imports them on the other side.
- Exports sync objects from the producer and consumer(s) and imports them on the other side.
- Buffers:
 - Exports consumer element attributes from downstream and imports them upstream.
 - Exports consolidated packet attribute list and packets from upstream and imports them downstream.
- > During streaming phase:
 - Source IPC block signals availability of new payloads to destination block, passing the fence.
 - Destination IPC block signals availability of packets for reuse to source block, passing the fence.
- > In non-safety builds, during streaming phase:
 - Signals changes in packet attributes and buffers from source to destination.



Note:

Reallocation of buffers in non-safety builds are not supported in this release.

6.6.4.4.5.2 Memory Boundary IPC

When the two processes do not share memory, memory boundary IPC blocks must be used. Each half of the stream must provide a separate set of packets. When new payloads arrive, the source block transmits all the data to the destination block, where it is copied into a new packet. Auxiliary communication channels may be set up for this purpose. Once transmission is done, the original packet is returned upstream for reuse, without waiting for the consumer to finish reading the data, since it accesses a different set of buffers.

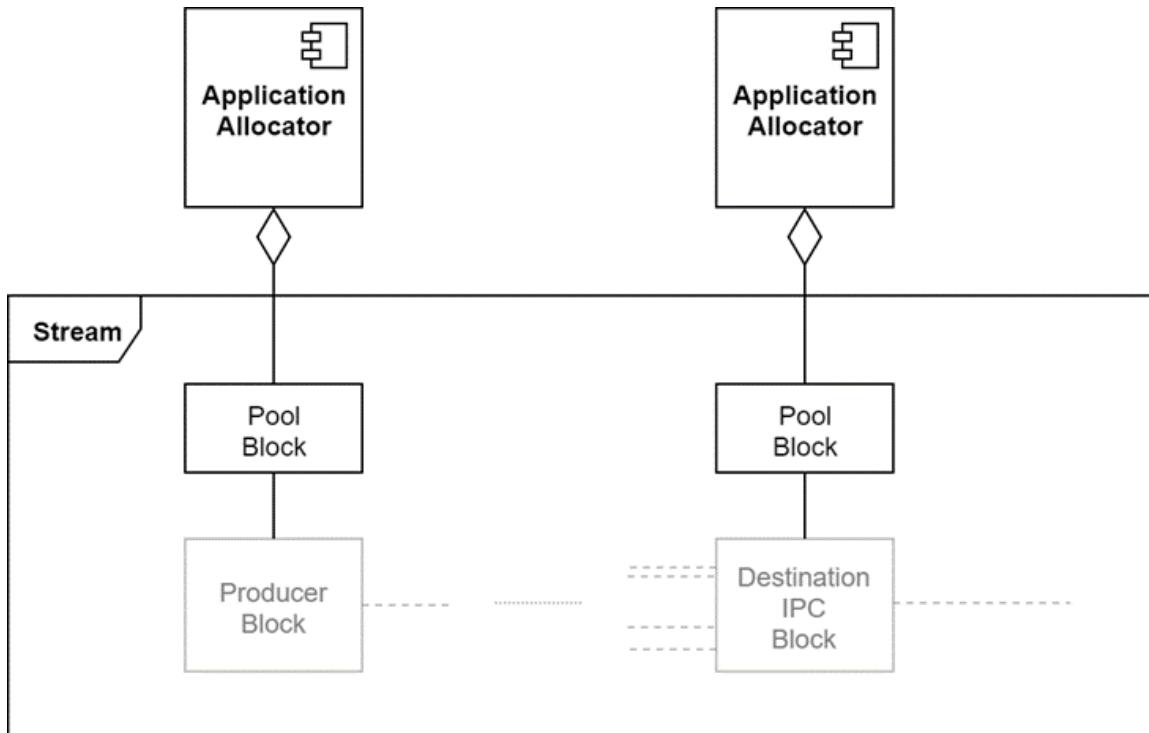
These IPC blocks can also be used to create virtual memory boundaries between portions of a stream. If one consumer operates at a lower level of safety and/or security than the rest of the stream, then even if it can share memory with the rest of the stream, it may not be desirable to do so. Requiring this consumer to use its own set of buffers ensures that if it fails, it will not prevent the rest of the stream from continuing, and if it falls prey to a security issue, it won't be able to modify the buffer data seen by the other consumers.

- > During setup phase:
 - Synchronization:
 - Synchronization objects are not exchanged across the memory boundary.
 - Source and destination blocks provide their synchronization requirements for the producer and consumer, respectively.
 - If necessary, creates sync objects to be used to coordinate the data copy, and passes to the local endpoint.
 - Accesses sync objects from the local endpoint to coordinate data copy.
 - Buffers:

- Exports a subset of the consumer element attributes from downstream and imports them upstream, replacing attributes related to memory access with those needed for the data transfer mechanism to access the memory.
 - Exports a subset of the consolidated packet attribute list from upstream and imports them downstream, again replacing attributes related to memory access with those needed for the copy engine.
 - On destination side, receives and maps buffers used for copy from downstream.
- > During streaming phase:
- Source IPC block transmits payload data to the destination block.
 - Destination IPC block reads the payload data into an available packet and passes it downstream.
- > In non-safety builds, during streaming phase:
- Signals changes in packet attributes and buffers from source to destination.

6.6.4.4.6 Pool

Pool blocks are used to introduce packets to the stream and track packets available for reuse. All streams must have at least one pool attached to the producer block. If a stream contains memory boundaries, then additional pools are needed for each section of the stream that uses its own set of packets, attached to the IPC block. For each pool block, there is an application component responsible for deciding the packet layout based on the requirements and allocating the buffers.



NvSciStream supports two kinds of pools: static and dynamic.

6.6.4.4.6.1 Static Pool

Static pools provide a set of packets that remain fixed for the life of the stream. The number of buffers must be specified at creation time, and the buffers are added during the setup phase. A static pool provides the following interactions with the application component that manages it:

- > During setup phase:
 - Buffers:
 - Reports list of element attributes producer can generate.
 - Reports list of element attributes consumers require.
 - Accepts consolidated packet attribute list, and sends to producer and consumers.
 - Accepts allocated packets and sends to producer and consumers.
- > During streaming phase:
 - Receives and queues packets returned to the producer for reuse.
 - Provides an available packet to the producer or IPC block it supports when requested.

6.6.4.4.6.2 Dynamic Pool

Dynamic pools are only available in non-safety builds. They allow buffers to be added and removed at any time. This supports use cases where the producer may need to change some of the buffer attributes, such as the size or pixel format of the data. Video playback is a typical example where such changes may occur. A dynamic pool provides all the interactions of a static pool, plus the following:

- > In non-safety builds, during streaming phase:
 - Reports requested element attribute changes from the producer.
 - Accepts updated element attributes and sends it to the producer and consumers.
 - Accepts instructions to remove packets from the stream and sends it to the producer and consumers.
 - Accepts new allocated packets and sends it to the producer and consumers.

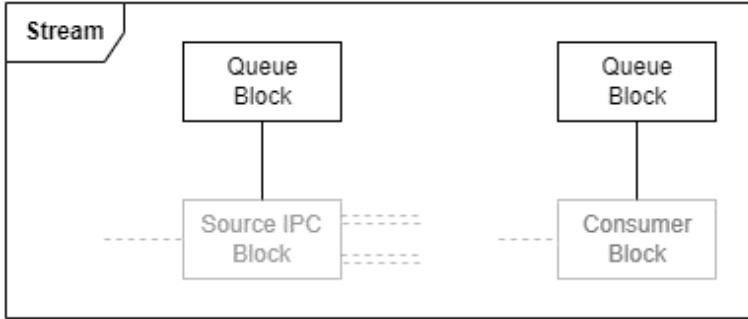


Note:

Dynamic blocks are not supported in this release.

6.6.4.4.7 Queue

Queue blocks attached to consumer blocks keep track of payloads waiting to be acquired by the consumer. Each consumer block must have an attached queue block to manage available packets. Queue blocks attached to memory boundary Source IPC blocks keep track of payloads waiting to be copied across memory boundary. Each memory boundary Source IPC block must have an attached queue block to manage available packets.



NvSciStream supports two types of queue blocks: FIFOs and mailboxes. A FIFO block is used when all data must be processed. Payloads always wait in FIFO until the consumer acquires them. Mailboxes are used when the consumer always acts on the most recent data. If a new payload arrives in the mailbox when one is already waiting, the previous one is skipped and immediately returned to the producer for reuse.

6.6.4.4.8 Limiter

A limiter block places a cap on the number of packets allowed to be sent downstream at any given time. It is usually inserted between a Multicast block and a Consumer block (and in general before any IPC blocks transmitting to the consumer).

If a new packet arrives and the current number of downstream packets is at the capacity, the packet is returned upstream immediately without reaching the consumer. This is similar to frame-skipping that can be invoked by a Mailbox queue block attached to the consumer. However, in this case the skipping can occur in the producer application rather than in the consumer. When a consumer at capacity returns a packet for reuse it can then receive a new packet.

A Limiter block does not interact with any other Limiter blocks used with other consumers. Each independently imposes its own limit on what is downstream and not affected by the operations of any other limiter.

In addition, when using a mailbox queue, the consumer application must be capable of dealing with skipped frames. Typical consumers may not be aware of skipped frames and operate on the frame they receive without regard to previous frames. Where consumers need to know that a frame is missed the application suite must include a sequence number, timestamp, or other identifying information in the packet data.

Since faulty consumers may take some packets out of service, the application may need to allocate additional total packets to ensure that there are enough left for the producer and remaining consumers to use if one consumer reaches its cap and never returns any packets again.

6.6.4.4.9 ReturnSync

A ReturnSync block addresses security and robustness with multicast streams. A ReturnSync block can be inserted anywhere in the stream between the Producer and Consumer blocks, but its primary intent is to be inserted between a Multicast block and a Consumer block, downstream of a Limiter block for that consumer. If a producer

application must consider one or more consumer applications generating unreliable fences, it can add ReturnSync blocks to safeguard against them. This isolates packets with bad fences in the branch for an untrusted consumer, which prevents the producer from getting stuck waiting for them.

A ReturnSync block waits for fences for each received packet from downstream before sending it upstream. If fences for the received packet are expired or empty, then it returns the packet upstream immediately. If not, it places the packet in the queue, and then triggers a thread spawned by this block to manage the waiting fence before sending the packet upstream. This block is useful if blocks upstream require synchronous access of packet data.

6.6.4.4.10 PresentSync

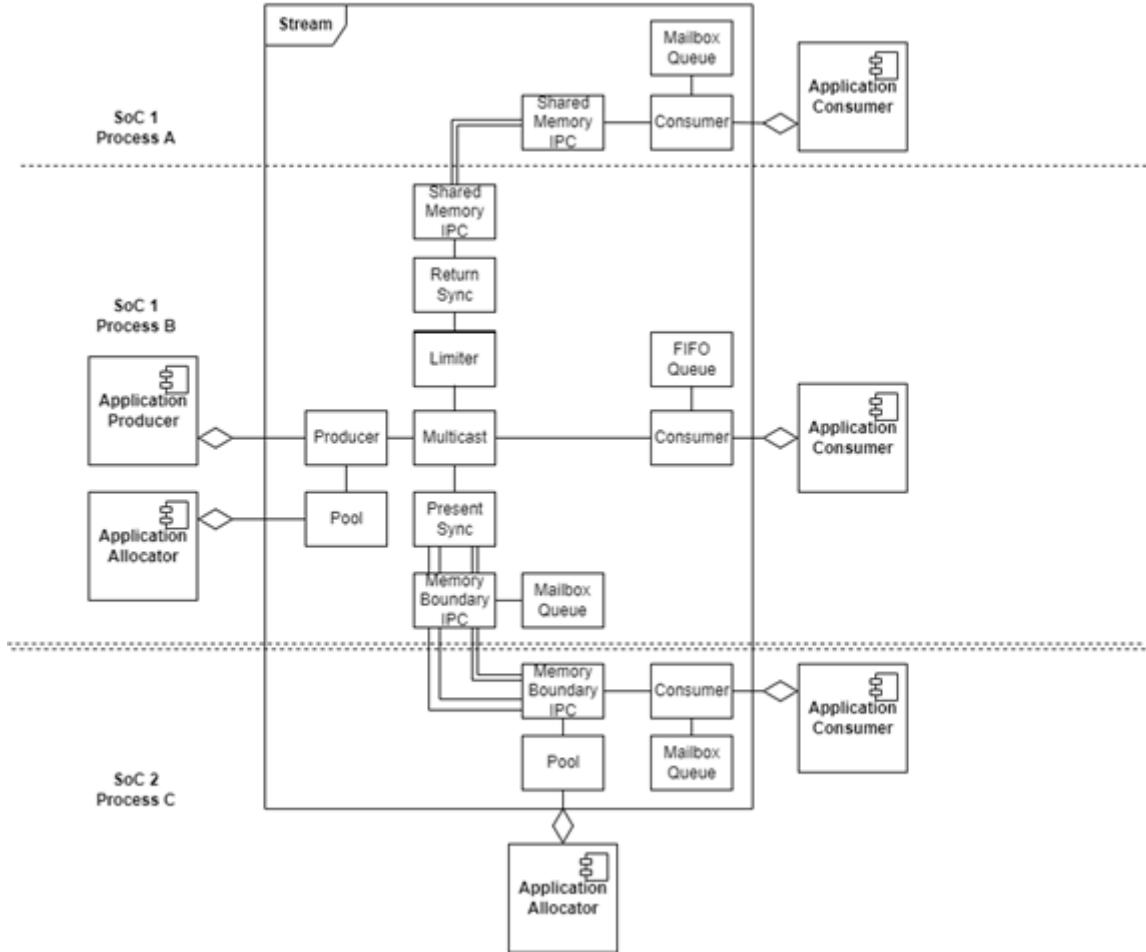
For symmetry, a PresentSync block is added, which does the waiting for fences for each received packet from upstream before sending it downstream. The block spawns a thread that waits for fences in the order of the packets received and send them downstream once fences are reached. This block is useful if blocks downstream require synchronous access of packet data.

For example, a PresentSync block can be inserted between the Producer and the memory boundary IPC block to keep packets from queuing for sending until their contents are ready; primarily when the memory boundary IPC block uses a mailbox queue.

6.6.4.4.11 Example

The following diagram shows a complete stream. It has three (3) consumers. One resides in the same process as the producer, the second resides in another process on the same system and shares memory with the producer process, and the third resides on another system and uses its own set of packets. The first uses a FIFO queue and the other two use mailbox queues. The producer application is concerned about the consumer in another process generating unreliable fences, it adds a return sync block to safeguard against it and places a cap on the number of packets held by the consumer in another process this consumer using the limiter block. It also chooses to add a present sync block before the memory boundary source IPC block, which uses a

mailbox queue, to keep packets from being queued for sending until their contents are



ready.

6.6.4.5 Stream Creation

The modular nature of NvSciStream allows producer and consumer components, or entire applications, to be developed independently by different providers, but some planning is required to ensure proper inter-operation. System integrators must make several decisions before assembling a stream.

- Determine which consumers reside in the same process as the producer and which are separated in other processes, partitions, or systems. Factors that influence this decision include modularity of development, management of computational resources, and freedom from interference.
- Determine which consumers require physical or virtual memory boundaries to safeguard critical consumers from less robust or secure consumers.
- Determine which consumers must process every payload and which only require the most recent data.
- Determine which consumers cannot be fully trusted and should have their packet access limited.
- Determine how communication between the processes is established.

- > Provide a uniform set of packet element types with associated data layout definitions for designing all endpoints.
- > Based on the complexity of the stream, determine how many packets are required to keep the pipeline operating at desired efficiency.

Once these decisions are made, use NvSciStream to assemble the desired stream(s).

6.6.4.5.1 NvSciBuf and NvSciSync Initialization

All NvSci buffer and sync object handles are associated with a particular NvSciBufModule and NvSciSyncModule, respectively. Any streaming application must begin by creating one of each of these modules. In general, an application may create more than one of each type of module, but there is rarely any reason to do so. However, all buffer and sync objects used with a given stream within a single process must be associated with the same modules. Different streams may use different modules.

6.6.4.5.2 NvScilpc Initialization

For cross-process streams, the applications must first establish communication with each other using NvScilpc. Refer to [Inter-Process Communication](#) for more information on how to set up the connections.

Once an NvScilpc channel is created, the applications may use it to do any required initial validation and coordination. Once that is complete, ensure no unprocessed messages remain in the channel, reset the channel, and pass the channel endpoint off to NvSciStream. NvSciStream then takes the ownership of the channel and uses it to coordinate stream operations between the two processes. From this point on, the application must not directly operate on the channel. Doing so leads to unpredictable behavior and will cause the stream to fail. After streaming is complete, the application should close the channel endpoint after deleting the NvSciStream blocks and freeing all the NvSciBuf/NvSciSync resources obtained from NvSciStream.

6.6.4.5.3 NvSciEventService Initialization

NvSciStream can optionally use NvSciEventService for event notification. For each NvSciStream block, applications can request an NvSciEventNotifier object on which applications can wait for new events in the block. Applications can also provide the NvSciEventService to handle internal NvIpc events and I/O messages on the IpcSrc and IpcDst blocks.

Refer to the [NvSciEventService API Usage](#) section on how to set up NvSciEventService and wait on an NvSciEventHandler object.

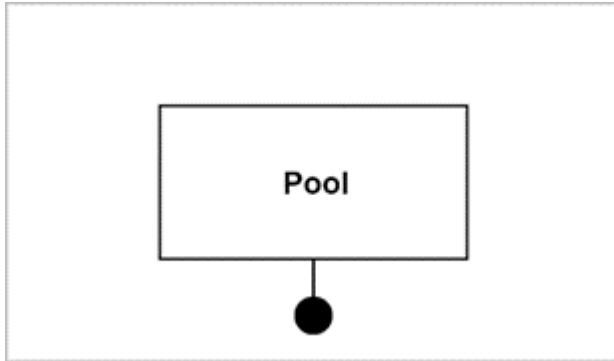
6.6.4.5.4 Block Creation

Each application is responsible for creating the NvSciStream blocks that reside in that process. The block creation functions all take a pointer parameter in. When successful, a block handle of type NvSciStreamBlock is returned. This handle is used for all further operations on the block. Some require additional parameters. For those that require a

NvSciBufModule and/or NvSciSyncModule, the same module must be provided as those used to access the buffers and sync objects. Blocks may be created in any order.

6.6.4.5.4.1 Pool

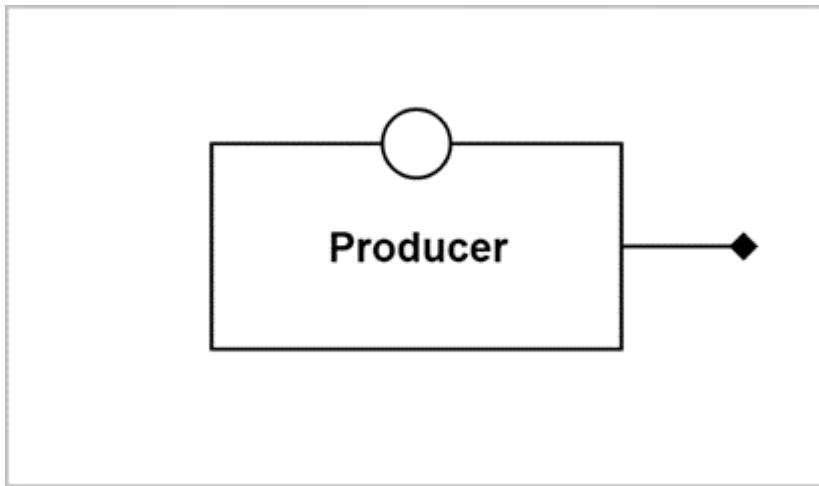
```
NvSciError
NvSciStreamStaticPoolCreate(
    uint32_t const      numPackets,
    NvSciStreamBlock *const pool
)
```



- Pool blocks have no inputs and outputs used with the connection function.
- Pools are attached directly to a producer or Memory Boundary IPC block during that block's creation.
- For static pools, the number of packets the pool provides must be specified at creation.

6.6.4.5.4.2 Producer

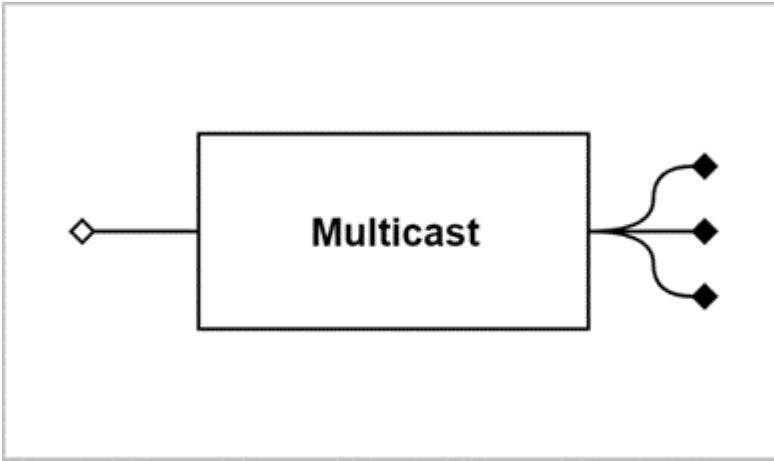
```
NvSciError
NvSciStreamProducerCreate(
    NvSciStreamBlock const pool,
    NvSciStreamBlock *const producer
)
```



- A pool block must be provided for each producer block at creation.
- Producer blocks have a single output connection and no input connections.

6.6.4.5.4.3 Multicast

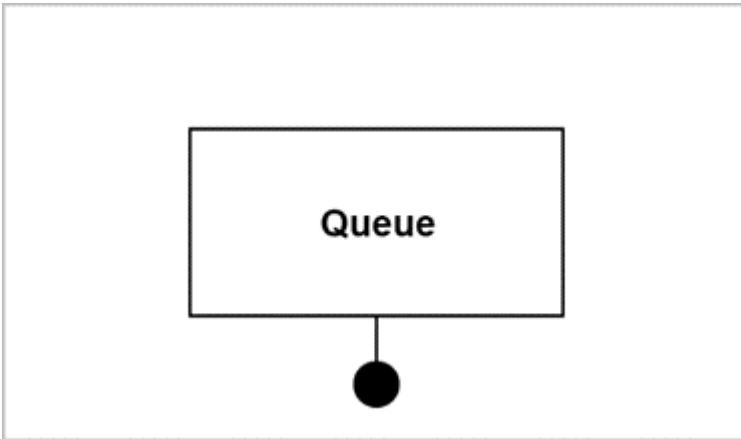
```
NvSciError
NvSciStreamMulticastCreate(
    uint32_t const          outputCount,
    NvSciStreamBlock *const multicast
)
```



- > Multicast blocks have a single input connection and a fixed number of output connections specified at creation.
- > During the connection process described in the following section, the order in which outputs are connected doesn't matter.

6.6.4.5.4.4 Queues

```
NvSciError
NvSciStreamFifoQueueCreate(
    NvSciStreamBlock *const queue
)
NvSciError
NvSciStreamMailboxQueueCreate(
    NvSciStreamBlock *const queue
)
```

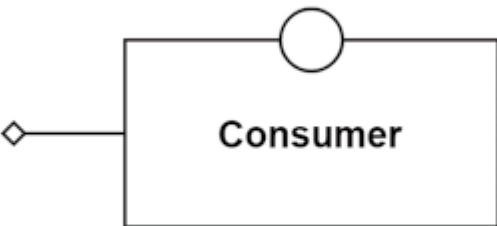


- > Queue blocks have no inputs and outputs used with the connection function.

- > Queues are attached directly to a consumer block or a memory boundary source IPC block during that block's creation.

6.6.4.5.4.5 Consumer

```
NvSciError
NvSciStreamConsumerCreate(
    NvSciStreamBlock const queue,
    NvSciStreamBlock *const consumer
)
```



- > A queue block must be provided for each consumer block at creation.
- > Consumer blocks have a single input connection and no output connections.

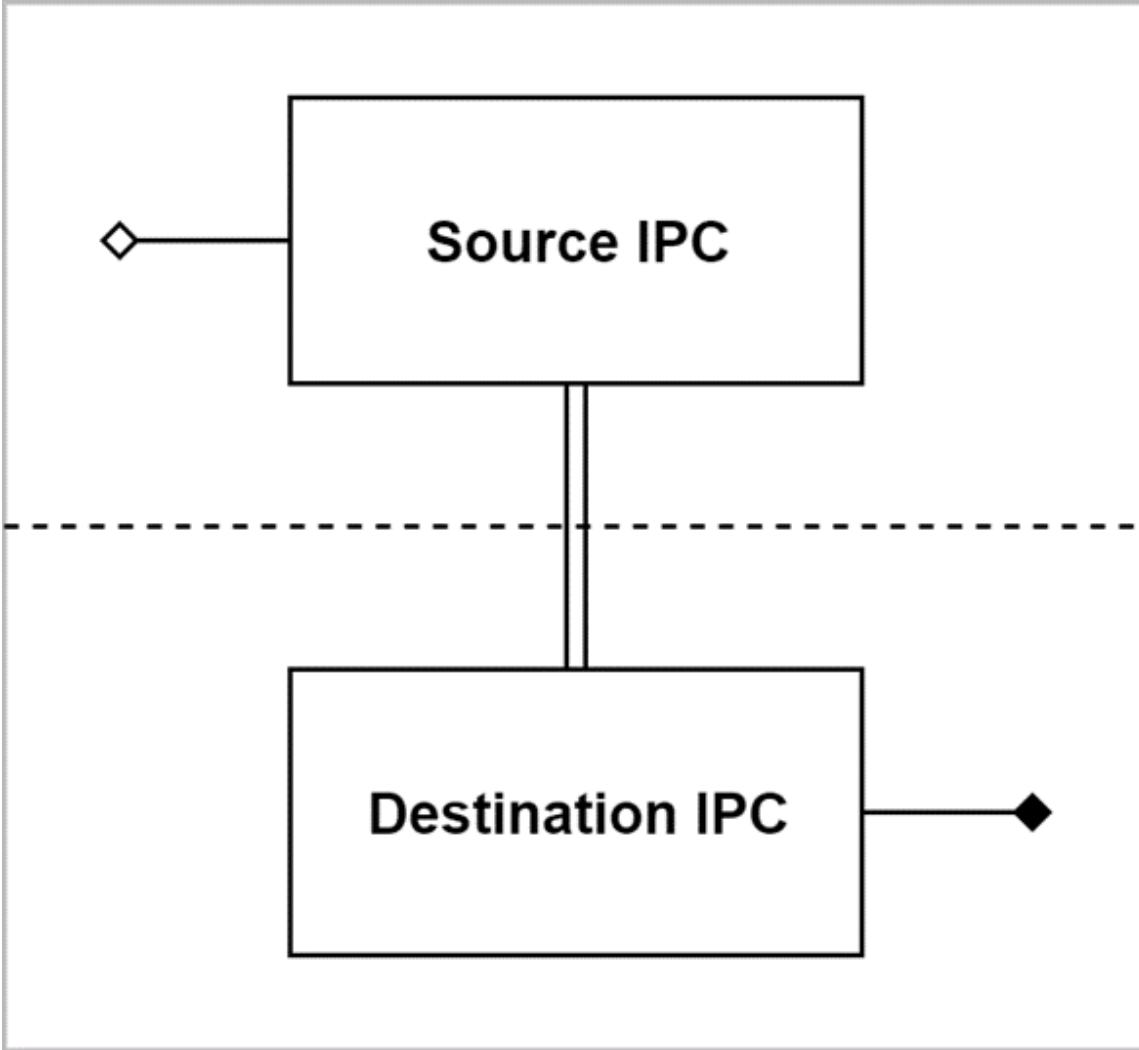
6.6.4.5.4.6 IPC

```
NvSciError
NvSciStreamIpcSrcCreate(
    NvSciIpcEndpoint const ipcEndpoint,
    NvSciSyncModule const syncModule,
    NvSciBufModule const bufModule,
    NvSciStreamBlock *const ipc
)

NvSciError
NvSciStreamIpcSrcCreate2(
    NvSciIpcEndpoint const ipcEndpoint,
    NvSciSyncModule const syncModule,
    NvSciBufModule const bufModule,
    NvSciStreamBlock const queue,
    NvSciStreamBlock *const ipc
)

NvSciError
NvSciStreamIpcDstCreate(
    NvSciIpcEndpoint const ipcEndpoint,
    NvSciSyncModule const syncModule,
    NvSciBufModule const bufModule,
    NvSciStreamBlock *const ipc
)
```

```
NvSciError  
NvSciStreamIpcDstCreate2(  
    NvSciIpcEndpoint const ipcEndpoint,  
    NvSciSyncModule const syncModule,  
    NvSciBufModule const bufModule,  
    NvSciStreamBlock const pool,  
    NvSciStreamBlock *const ipc  
)
```



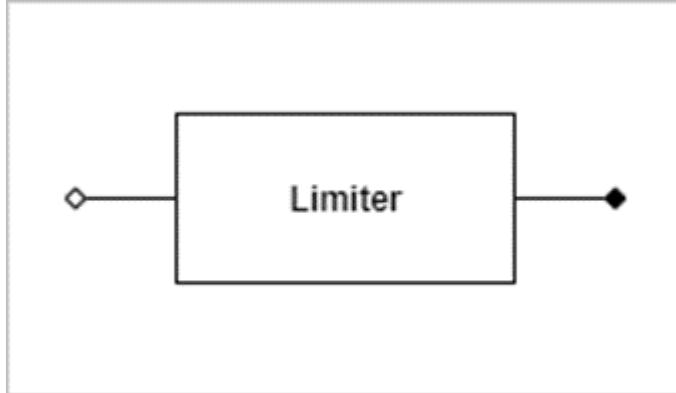
- IPC blocks are created in pairs, each using one end of an NvSciIpc channel.
- The caller must complete any of its own communication over the channel before passing it to NvSciStream and must not subsequently read from or write to it.
- The NvSciBuf and NvSciSync modules must be those used for all buffer and sync object associated with the stream in the calling process. The block uses them when importing objects from the other endpoint.
- The source (upstream) block has one input connection, and the destination (downstream) block has one output connection. Together with the channel between

them, they are viewed as a single virtual block with one input and one output, spanning the two processes.

6.6.4.5.4.7 Limiter

```
NvSciError
NvSciStreamLimiterCreate(
    uint32_t const      maxPackets,
    NvSciStreamBlock *const limiter
)
```

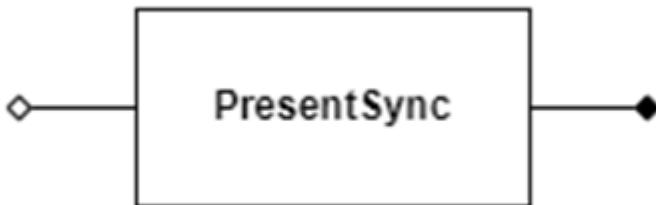
>



- > Limiter blocks have one input and one output.
- > The number of packets allowed to be sent downstream to a consumer block is specified at creation.

6.6.4.5.4.8 PresentSync

```
NvSciError
NvSciStreamPresentSyncCreate(
    NvSciSyncModule const syncModule,
    NvSciStreamBlock *const presentSync
)
```

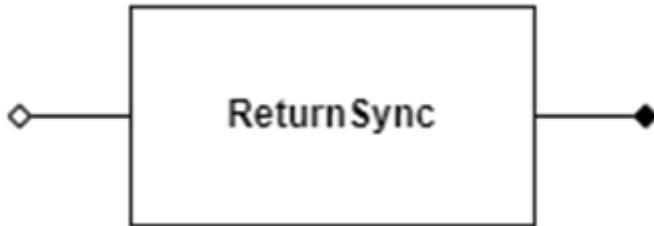


- > PresentSync block has one input and one output.
- > The NvSciSyncModule used to create the block must be the module endpoints use to allocate NvSciSyncObj.

6.6.4.5.4.9 ReturnSync

```
NvSciError
NvSciStreamReturnSyncCreate(
    NvSciSyncModule const syncModule,
```

```
NvSciStreamBlock *const returnSync
)
```



- > ReturnSync block has one input and one output.
- > The NvSciSyncModule used to create the block must be the module endpoints use to allocate NvSciSyncObj.

6.6.4.5.5 Configure Block to Use NvSciEventService

As an optional setup, applications can configure a block to use NvSciEventService for event notification.

```
NvSciError
NvSciStreamBlockEventServiceSetup(
    NvSciStreamBlock const block,
    NvSciEventService *const eventService,
    NvSciEventNotifier **const eventNotifier
)
```

The function returns a NvSciEventNotifier object that applications can use to wait for new events on the block. It is the responsibility of the application to delete the NvSciEventNotifier objects when they are no longer needed. Event-notification behavior of a block is undefined after its associated NvSciEventNotifier object is deleted.

If any NvSciStream API, including block connection, is called on a block before this function, the block will automatically be configured to use the default event-notification method, described in the following Event Handling section. The event-notification method of a block cannot be changed once it is determined.

6.6.4.5.6 Configure NvSciStreamBlock to Use External NvSciEventService for Internal Events

The application can configure the block for the user-provided NvSciEventService to wait for internal event notifications and trigger the corresponding handlers.

```
NvSciError
NvSciStreamBlockInternalEventServiceSetup(
    NvSciStreamBlock const block,
    NvSciEventService* const eventService,
    uint32_t* const notifierCount,
    NvSciEventNotifier** const eventNotifierArray
);
```

This function binds the internal event notifiers to the user-provided NvSciEventService and registers the NvSciStream internal handlers for each internal event notifier. The application must wait for events on these returned NvSciEventNotifiers. When there is a new event on the notifier, the corresponding handler is triggered automatically to process the NvSciStream internal event. These handlers are registered by NvSciStream internally and the application does not need to assign handlers to these notifiers.

When called on a IpcSrc or IpcDst block, the notifiers are associated with internal NvIpc events and messages. The internal handler processes message I/O.

If this function is not called before the block connection, the IpcSrc or IpcDst block automatically creates an internal NvSciEventService and spawns a dispatch thread to wait for and handle the internal events.

The application is responsible for destroying the NvSciEventNotifier and NvSciEventService when it is no longer needed. The event-notification behavior of a block is undefined after its associated NvSciEventNotifier object is deleted.

6.6.4.5.7 User-defined Endpoint Information

As an optional setup, before the blocks are connected, applications can supply user-defined data to the endpoint blocks (Producer and Consumer) with NvSciStreamBlockUserInfoSet(). After the blocks are connected, applications can query the data from every block in the connected stream with NvSciStreamBlockUserInfoGet(). All applications that operate the blocks should understand the userType value. Endpoint applications can use the information from opposed endpoints to make choices in resource creation or streaming.

```
NvSciError
NvSciStreamBlockUserInfoSet(
    NvSciStreamBlock const block,
    uint32_t const userType,
    uint32_t const dataSize,
    void const* const data
)

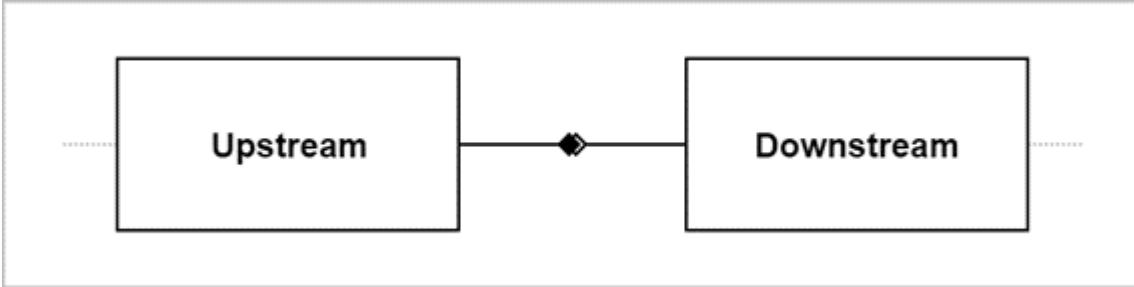
NvSciError
NvSciStreamBlockUserInfoGet(
    NvSciStreamBlock const block,
    NvSciStreamBlockType const queryBlockType,
    uint32_t const queryBlockIndex,
    uint32_t const userType,
    uint32_t* const dataSize,
    void* const data
)
```

6.6.4.5.8 Block Connection

Once blocks are created, they can be connected in pairs. The connection can be done in any order and can be intermingled with creation of the blocks.

```
NvSciError
```

```
NvSciStreamBlockConnect(
    NvSciStreamBlock const upstream,
    NvSciStreamBlock const downstream
)
```



An available output of the upstream block is connected to an available input of the downstream block. If there is no available output or input, the function fails. Each input and output block can be connected only once. A new block cannot be connected in place of an old one, even if the old one is destroyed.

6.6.4.5.9 Comparison with EGL

Those familiar with EGL will notice that this process is more involved than the creation of an EGLStream. For an EGLStream, all the desired features are encoded into an array of attributes, and then a single stream object (or two in the case of cross-process) is created. The details of setting up all the stream management are left to the EGL implementation. But a simple attribute array cannot convey all the possible stream feature permutations and use cases that an application may desire. In fact, multicasting to more than one consumer cannot be handled by EGLStream at all without additional extensions defining new objects.

The modular approach used by NvSciStream requires applications to perform separate creation calls for each feature of the desired stream, and then connect them together. This is more effort, but affords greater control over the stream's behavior, and allows for features not originally anticipated or readily supported by EGLStreams, such as multicasting. It allows the development of new block types to support new features in the future, which are harder to add to a monolithic stream object.

6.6.4.5.10 Simple Example

The following example assembles a simple cross-process stream with a producer and pool in one process, and a FIFO and consumer in another process. The cross-process communication channel is assumed to be established before this code executes.

6.6.4.5.10.1 Sample Producer Creation

```
// We'll use triple buffering for this stream
const uint32_t numPackets = 3;
// Stream variables
NvSciIpcEndpoint      srcIpc;
NvSciBufModule         bufModule      = 0;
NvSciSyncModule        syncModule     = 0;
NvSciStreamBlock       producerBlock = 0;
```

```

NvSciStreamBlock    poolBlock      = 0;
NvSciStreamBlock    srcIpcBlock   = 0;
NvSciError         err;
// Setting up communication is outside the scope of this guide
srcIpc = <something>;
// Set up buffer and sync modules
// (If using multiple streams, or doing other non-stream NvSci
// operations, these might be passed in from some global setup.)
err = NvSciBufModuleOpen(&bufModule);
if (NvSciError_Success != err) {
    <handle failure>
}
err = NvSciSyncModuleOpen(&syncModule);
if (NvSciError_Success != err) {
    <handle failure>
}
// Create all the stream blocks
err = NvSciStreamStaticPoolCreate(numPackets, &poolBlock);
if (NvSciError_Success != err) {
    <handle failure>
}
err = NvSciStreamProducerCreate(poolBlock, &producerBlock);
if (NvSciError_Success != err) {
    <handle failure>
}
err = NvSciStreamIpcSrcCreate(srcIpc, syncModule, bufModule, &srcIpcBlock);
if (NvSciError_Success != err) {
    <handle failure>
}
// Connect the blocks
err = NvSciStreamBlockConnect(producerBlock, srcIpcBlock);
if (NvSciError_Success != err) {
    <handle failure>
}

```

6.6.4.5.10.2 Sample Consumer Creation

```

// Stream variables
NvSciIpcEndpoint    dstIpc;
NvSciBufModule       bufModule     = 0;
NvSciSyncModule      syncModule   = 0;
NvSciStreamBlock     consumerBlock = 0;
NvSciStreamBlock     fifoBlock    = 0;
NvSciStreamBlock     dstIpcBlock  = 0;
NvSciError          err;
// Setting up communication is outside the scope of this guide
dstIpc = <something>;
// Set up buffer and sync modules
// (If using multiple streams, or doing other non-stream NvSci
// operations, these might be passed in from some global setup.)
err = NvSciBufModuleOpen(&bufModule);
if (NvSciError_Success != err) {
    <handle failure>
}

```

```

err = NvSciSyncModuleOpen(&syncModule);
if (NvSciError_Success != err) {
    <handle failure>
}
// Create all the stream blocks
err = NvSciStreamFifoQueueCreate(&fifoBlock);
if (NvSciError_Success != err) {
    <handle failure>
}
err = NvSciStreamConsumerCreate(fifoBlock, &consumerBlock);
if (NvSciError_Success != err) {
    <handle failure>
}
err = NvSciStreamIpcDstCreate(dstIpc, syncModule, bufModule, &dstIpcBlock);
if (NvSciError_Success != err) {
    <handle failure>
}
// Connect the blocks
err = NvSciStreamBlockConnect(dstIpcBlock, consumerBlock);
if (NvSciError_Success != err) {
    <handle failure>
}

```

6.6.4.6 Event Handling

NvSciStream is designed so that, once a stream is created and connected, applications can follow an event-driven model. Operations on one block in a stream trigger events in other blocks. These events are dequeued by the applications, which act in response, performing new block operations that trigger new events, and so on. Every block supports an event queue, although some types of blocks may only generate events during the initial setup phase.

6.6.4.6.1 Event Query

Events can be queried from each block:

```

NvSciError
NvSciStreamBlockEventQuery(
    NvSciStreamBlock const      block,
    int64_t const              timeoutUsec,
    NvSciStreamEventType *const eventType
)

```

- > If the block is not configured to use NvSciEventService:
- > • If no events are pending in the block's queue, a non-zero timeout causes it to wait the specified number of microseconds for an event to arrive. If a negative value is used, the call waits forever.
- > If the block is configured to use NvSciEventService:
- > • Non-zero timeout causes it to return an NvSciError_BadParameter error.

On success, a pending event is removed from the block's queue, the output parameter is filled with the type of the event, and `NvSciError_Success` is returned. If no event is available before the timeout period ends, an `NvSciError_Timeout` error is returned.

After an event is queried from a block, certain `NvSciStream` APIs become operational on the block. Refer to the API Reference for `NvSciStreamEventType` for additionally information about the functions that become operational for each event type queried.

6.6.4.6.2 Event Notification

`NvSciStream` provides two event-notification methods. This is a per-block configuration, and it is not required that all blocks in a stream be configured to use the same event-notification method.

- > Without `NvSciEventService`

Applications wait for events by calling `NvSciStreamBlockEventQuery()` with non-zero timeout value.

- > With `NvSciEventService`

Applications wait for events by waiting on the `NvSciEventNotifier` objects bound to the blocks, using the event-waiting API of `NvSciEventService`. Applications should query all the events pending in the block after waking up from waiting on the associated `NvSciEventNotifier` object. Note it is possible that `NvSciEventNotifier` objects be signaled by spurious events, in which case the event query will return `NvSciError_Timeout` error.

The event-waiting API of `NvSciEventService` waits on `NvSciEventNotifier` objects, regardless of the UMD they are associated. It enables applications to simultaneously wait for `NvSciStream` events and events from other UMDs.

Refer to the [NvSciEventService API Usage](#) section for how to wait on a single or multiple `NvSciEventNotifier` objects.

6.6.4.6.3 Connection and Disconnection Events

Assuming no failures occur during setup, the first event type each block receives is `NvSciStreamEventType_Connected`. This indicates that the block has a complete connection path to the producer and consumers, respectively. After connecting, no operations are allowed on any block until this event is received. Applications must wait for it before proceeding with the resource setup described in the next section.

If a block is destroyed or, in the cross-process case, communication with a process is lost, an `NvSciStreamEventType_Disconnected` event is sent to connected blocks. If other events are pending, such as available payloads, they are processed first. Once this event is received, no further events arrive, and operations to send events fail.



Note:

In safety-certified systems, failures and teardown are never supposed to occur. Although disconnect events are currently supported in the current safety release, they will be removed in later safety releases, and only supported for non-safety platforms.

6.6.4.6.4 Error Events

If the event queried is of type NvSciStreamEventType_Error, an error not directly triggered by user action occurred in the block. Applications can retrieve the error code by NvSciStreamBlockErrorGet(). If multiple errors occur before this retrieval, only the first error code is available. All subsequent errors are ignored until the error code is retrieved, with the assumption that they are related to the first error.

```
NvSciError
NvSciStreamBlockErrorGet(
    NvSciStreamBlock const block,
    NvSciError* const status
)
```

For PCIe errors, such as NvSciError_PcieUncorrectableFatal, NvSciError_PcieUncorrectableNonFatal, and NvSciError_PcieEdmaTransferErr, the application must ensure NvSciIpcEndpoint is in a usable state before recreating the NvSciStream pipeline.

6.6.4.7 Resource Creation

Once the stream blocks are fully created, the next step is to create synchronization and buffer resources. The process of determining resource requirements and allocating them is much like that described in the simple single-buffer example at the beginning of this chapter. But all coordination between the producer and consumers is done through the stream, which automatically deals with any translations required to share the resources between processes, partitions, or systems.

Creation of synchronization and buffer resources can be done in either order or can be intermingled. The two are similar, but the process for synchronizing objects is a little simpler.

Progression of Resource Creation

Various setup operations are divided into key groups. Data for each group is gathered and sent together at once when the application indicates that it is done with the setup. This completion is signaled with a call to NvSciStreamBlockSetupStatusSet(). The completed parameter to this function is for future support of dynamically modifying streams (for non-safety usecases) and currently must always be true.

```
NvSciError
NvSciStreamBlockSetupStatusSet(
    NvSciStreamBlock const block,
    NvSciStreamSetup const setupType,
    bool const completed
)
```

6.6.4.7.1 Buffer Resources

To allocate buffers, the producer and consumer blocks communicate their requirements to a third block, the pool. The application component that owns this pool block is responsible for performing the allocations. For many use cases, this is the same

application that manages the producer. But keeping this functionality in a separate block from the producer serves several purposes.

It allows NvSciStream to provide different types of pools for different use cases while sharing a common set of interfaces. For instance, you may choose a static pool with a fixed set of buffers for safety-certified builds, a dynamic pool that allows buffers to be added and removed in cases where the producers may change the data layout over time, or a remote pool managed by a central server process that doles out buffers to all streams in the application suite.

Furthermore, a stream may require additional pools beyond the one which feeds buffers to the producer. For instance, in cross-system use cases where memory is not shared between the producer and consumer, the IPC block on the consumer side also requires a pool to provide buffers into which data is copied. Keeping the pool as a separate block allows generic application components to be written that do not need to know whether the pool is used with a producer or another block.

Before reading this section, refer to [Buffer Management](#) to understand how buffer requirements are specified and how buffers are created. This section assumes familiarity with the commands used there and does not explain them in detail. This section covers how to coordinate buffers through a stream.

6.6.4.7.1.1 Buffer Requirements

Specifying Requirements

Buffer data flows from the producer to the consumer(s). The producer must query the NVIDIA drivers they use for buffer attribute lists that provide write capability, while consumers must query for buffer attributes that provide read capability. If the buffer memory is written or read directly with the CPU, attribute lists can be manually created, requesting CPU access.

A packet may consist of multiple buffer elements containing different types of data. The producer must obtain a buffer attribute list for each type of data it can generate, and consumers must obtain attribute lists for each type of data they require.

Once a producer knows all the elements it can provide, or a consumer knows all the elements it requires, it can inform the stream by calling `NvSciStreamBlockElementAttrSet()` (operational after `NvSciStreamEventType_Connected` is queried) for each element. The `userType` parameter is an application-defined value to identify the element, understood by both the producer and consumer. The `userType` should be unique for each element. The `bufAttrList` field contains a handle for the element's attribute list. Ownership of this handle remains with the caller, and it deletes it when it is no longer needed after the function returns. NvSciStream creates a duplicate.

After specifying all of the elements, the application indicates it finished element setup by calling `NvSciStreamBlockSetupStatusSet()` with a value of `NvSciStreamSetup_ElementExport`. NvSciStream automatically determines the indices and count.

Asynchronous elements data is not available to the consumer until the fences complete. This generally indicates data written using NVIDIA hardware. Synchronous elements data is available to be read as soon as the packet is acquired by the consumer, without waiting for a fence. This generally indicates data written directly with the CPU. Sync requirements setup steps determine whether an element is asynchronous or synchronous.

A typical use case is that a packet contains both asynchronous and synchronous elements: a packet containing a large primary data element accompanied by a smaller metadata element. The smaller element contains information to program the hardware for processing the primary element. Upon acquiring the packet, the CPU must read the metadata immediately so that instructions for the NVIDIA drivers can be issued. The primary data is read later once the fence has been reached.

```
NvSciError
NvSciStreamBlockElementAttrSet(
    NvSciStreamBlock const block,
    uint32_t        const userType,
    NvSciBufAttrList const bufAttrList
)
```

Receiving Requirements

Pool, producer, and consumer each receive a single `NvSciStreamEvent_Elements` event. The pool receives the event after the producer and all consumers finish specifying their element support. After the pool finishes specifying the final packet element layout, the producer and consumers receive this event. After this event is queried from pool, the following functions become operational on the pool block:

```
NvSciStreamBlockElementCountGet()
NvSciStreamBlockElementAttrGet()
NvSciStreamBlockElementAttrSet()
```

After the producer and consumers query the event, the following functions become operational on the producer and consumer blocks:

```
NvSciStreamBlockElementCountGet()
NvSciStreamBlockElementAttrGet()
NvSciStreamBlockElementUsageSet() (consumer only)
NvSciStreamBlockElementWaiterAttrSet()
```

As with synchronization requirements, the attributes received by the pool may not exactly match those sent by the producer and consumer endpoints. For the multicast case, the pool receives a combined list with one element for each type of attribute the consumers requested. Elements with the same type arrive as a single event with their attribute lists merged. The stream may also transform attributes to handle cross-process or cross-system cases. When querying the elements, consumers can choose not to use all of them. A consumer can inform `NvSciStream` that an element will not be used by calling `NvSciStreamBlockElementUsageSet()` with `<used>` set to false. This will allow `NvSciStream` to optimize by not sharing the relevant buffers with the consumer. This function can be called with `<used>` set to true, but this is the default, and the call is not necessary. Therefore, most existing applications will not need to add this call.

After querying element information and (for the consumer) indicating which elements they will support, the producer and consumer(s) must call `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_ElementImport`. This allows them to begin receiving packets from the pool.

Any secondary pools connected to IPC blocks also receive events for the producer and consumer elements. However, the producer events are delayed until the primary pool determines the final packet layout (discussed below). The producer element attributes the secondary pools receive are those sent by the primary pool.

```

NvSciError
NvSciStreamBlockElementCountGet(
    NvSciStreamBlock const block,
    NvSciStreamBlockType const queryBlockType,
    uint32_t* const numElements
)

NvSciError
NvSciStreamBlockElementAttrGet(
    NvSciStreamBlock const block,
    NvSciStreamBlockType const queryBlockType,
    uint32_t const elemIndex,
    uint32_t* const userType,
    NvSciBufAttrList* const bufAttrList
)

NvSciError
NvSciStreamBlockElementUsageSet(
    NvSciStreamBlock const block,
    uint32_t const elemIndex,
    bool const used
)

```

Reconciling Requirements

For the pool attached to the producer:

The application managing the pool must take the capabilities provided by the producer and the requirements provided by the consumer(s) and determine the final packet layout. For each element type provided by the producer and required by a consumer, it should merge their attribute lists to obtain the attribute list to use in allocating the buffer. If there is an element type provided by the producer but not required by any of the consumers, omit it from the packets. If there is a type required by a consumer that the producer cannot provide, the application can trigger an error. However, there may be cases understood by the application suite where the consumer requirements represent several options, and only one of the requested element types is needed. If the producer can provide one of them, streaming can proceed. It is to support possible complex situations like this that the reconciliation process is left to applications, rather than being done automatically by the stream.

As in producer and consumer(s) applications, pool application calls the `NvSciStreamBlockSetupStatusSet()` function with a value of

`NvSciStreamSetup_ElementImport` to inform NvSciStream it has finished element import.

Once the application has determined the final layout, it calls `NvSciStreamBlockElementAttrSet()` to inform NvSciStream the reconciled element attributes. It also must call `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_ElementExport` to inform NvSciStream it finished exporting element attributes. Producer and consumer(s) application(s) query event `NvSciStreamEventType_Elements` and get the element attributes by the function described above. Applications use this information to interpret the data layout and prepare to receive the packets.

For the pool attached to the memory-boundary destination IPC:

The application retrieves the element attributes reconciled by the producer's pool after querying event `NvSciStreamEventType_Elements`. The application calls the `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_ElementImport` to inform NvSciStream it finished element import.

6.6.4.7.1.2 Buffer Exchange

Specifying Buffers

Once the element attributes are specified, the application can allocate buffers from `NvSciBuf` using the attribute lists. It creates packets using the pool object, allocates a buffer for each element of each packet, and then assigns them to their proper places in the packets.

To create a new packet, the application calls `NvSciStreamPoolPacketCreate()`. The cookie is any non-zero value that the application chooses to look up data structures for the packet. Typically, the cookie is either a 1-based index into an array of structures or a pointer directly to the structure. Any events received on the pool object related to the packet references this cookie. On success, the function returns a new handle that the application stores in its data structure for the packet and uses whenever it needs to tell the stream to operate on the packet.

After creating a packet, the application assigns a buffer to each element by calling `NvSciStreamPoolPacketInsertBuffer()`. The packet handle is returned at packet creation, and the index of the element is assigned the buffer handle, which NvSciStream duplicates. Ownership of the original remains with the caller, and once the function returns, it may safely free the buffer.

The application calls `NvSciStreamPacketComplete()` to inform NvSciStream finished assigning buffers for a packet.

When the application finished creating all packets and assigned all buffers, it calls the `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_PacketExport` to indicate completion of packet creation.

For static pools, the number of packets is specified when the pool is created. Streaming won't start until this number of packets is created. If the application tries to create more than this number of packets, an error occurs.

```
NvSciError
NvSciStreamPoolPacketCreate(
    NvSciStreamBlock const pool,
    NvSciStreamCookie const cookie,
    NvSciStreamPacket *const handle
)

NvSciError
NvSciStreamPoolPacketInsertBuffer(
    NvSciStreamBlock const pool,
    NvSciStreamPacket const handle,
    uint32_t const index,
    NvSciBufObj const bufObj
)

NvSciError
NvSciStreamPoolPacketComplete(
    NvSciStreamBlock const pool,
    NvSciStreamPacket const handle
)
```

Receiving Buffers

When a pool completes a new packet, any producer or consumers that pool serves is notified with a `NvSciStreamEventType_PacketCreate` event.

Upon receiving this event, the endpoint calls `NvSciStreamBlockPacketNewHandleGet()` to dequeue the handle of the new packet and then

`NvSciStreamBlockPacketBufferGet()` to get the buffers for the elements in the packet. After checking whether it can map in all the buffers for a given packet, the endpoint signals status back to the pool by `NvSciStreamBlockPacketStatusSet()`. The status parameter indicates if the application was successful in setting up the new packet. If so, the value is `NvSciError_Success`. Otherwise, it can be any value the application chooses. `NvSciStream` does not interpret the value except to check for success, and then passes it back to the pool. If successful, the cookie parameter provides the endpoint cookie for the packet. Each endpoint can provide its own cookie for each packet, which is used in subsequent events.

The producer and consumers may assign the same cookies as the pool but are not required to do so.

```
NvSciError
NvSciStreamBlockPacketNewHandleGet(
    NvSciStreamBlock const block,
    NvSciStreamPacket* const handle
)

NvSciError
NvSciStreamBlockPacketBufferGet(
    NvSciStreamBlock const block,
```

```

    NvSciStreamPacket const handle,
    uint32_t const elemIndex,
    NvSciBufObj* const bufObj
)

NvSciError
NvSciStreamBlockPacketStatusSet(
    NvSciStreamBlock const block,
    NvSciStreamPacket const handle,
    NvSciStreamCookie const cookie,
    NvSciError const status

```

After the pool finishes exporting the packets, the endpoints receive an `NvSciStreamEventType_PacketsComplete` event. They can complete setup related to packet resources and call the `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_PacketImport` to indicate they finished importing the packets.

If a pool deletes a packet, when the producer or consumer receives the `NvSciStreamEventType_PacketDelete` event it can determine the identity of the deleted packet by calling `NvSciStreamBlockPacketOldCookieGet()`, which retrieves the cookie of a packet pending deletion. The handle of the returned packet becomes invalid for subsequent function calls.

```

NvSciError
NvSciStreamBlockPacketOldCookieGet(
    NvSciStreamBlock const block,
    NvSciStreamCookie* const cookie
)

```

Completing Buffer Setup

When the producer and consumers accept a packet, a the pool receives the `NvSciStreamEventType_PacketStatus` event. The pool application calls `NvSciStreamPoolPacketStatusAcceptGet()` for the acceptance status of a packet. If the packet is not accepted by producer or consumers, the pool application can call `NvSciStreamPoolPacketStatusValueGet()` to get the error code.

After receiving the acceptance status of all packets, the pool application calls the `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_PacketImport` to complete the buffer setup.

```

NvSciError
NvSciStreamPoolPacketStatusAcceptGet(
    NvSciStreamBlock const pool,
    NvSciStreamPacket const handle,
    bool* const accepted
)

NvSciError
NvSciStreamPoolPacketStatusValueGet(
    NvSciStreamBlock const pool,
    NvSciStreamPacket const handle,
    NvSciStreamBlockType const queryBlockType,
    uint32_t const queryBlockIndex,

```

```
NvSciError* const status
)
```

6.6.4.7.1.3 Comparison with EGL

When using EGLStreams, back-and-forth coordination of buffer attributes between the endpoints is not required. Buffers are created through the producer rendering interface and are communicated to the consumer when inserted in the stream. The downside is that, in most cases, the producer has no awareness of the consumer for which the buffers are intended. There is no way to ensure that the buffers the producer allocates are compatible with the consumer at the time they are created. If they are not, then either streaming fails when the consumer receives the buffers, or a costly conversion process must occur for every frame. The NvSciStream model puts more burden on the application when establishing the buffers but ensures optimal allocation settings for compatibility between producer and consumers.

Additionally, EGLStreams are far more restricted in the kinds of buffers they support. They allow two-dimensional image buffers rendered by the GPU and video, along with a limited set of metadata buffers generated by the CPU. NvSciStream allows multiple buffers of data rendered by either source. These buffers can contain images, arrays, tensors, or anything else the user requires.

6.6.4.7.2 Synchronization Resources

Before reading this section, be sure to read the full chapter on [Synchronization](#) to understand how synchronization requirements are specified and synchronization objects are created. This section assumes familiarity with the commands used there and does not explain them in detail. The following sections cover how to coordinate synchronization objects through a stream.

6.6.4.7.2.1 Synchronization Requirements

Setting up synchronization resources begins by determining the requirements of each endpoint. The producer and consumers must query the NVIDIA drivers they are going to use with the appropriate APIs to obtain separate NvSciSyncAttrList handles representing the requirements for signaling (writing to) and waiting for (reading from) synchronization objects.

If an endpoint directly writes to or reads from the stream packets with the CPU instead of using an NVIDIA API, it does not generate any fences, and therefore does not need synchronization signaling requirements. It needs to perform CPU waits for fences from the other endpoint, and therefore must create a waiting requirement attribute list with the NeedCpuAccess flag set.

In some use cases, an application endpoint may require that packets it receives be available immediately, without waiting for any fence. In this case, it does not need to provide any synchronization waiting requirement attributes. Instead, it indicates to the other endpoint that it must do a CPU wait before sending the packets. This is not common but is provided to support these cases when they arise.

Once an endpoint has determined its signal and wait requirements for synchronization objects, it stores the signal requirements locally, and passes the

wait requirements, one NvSciSyncAttrList per packet element, to the other endpoint through the stream. It passes an NvSciSyncAttrList with the wait requirement attribute if fences are supported into the endpoint block by calling NvSciStreamBlockElementWaiterAttrSet(). If the waiter for the element referenced by elemIndex does not support fences, NULL pointer is passed. The function call is the same for both producer and consumer endpoints. Ownership of the sync attributes' handle remains with the caller. The stream creates a duplicate before the function returns.

After indicating their waiter requirements for all the elements, the producer and consumer applications call the NvSciStreamBlockSetupStatusSet() function with a value of NvSciStreamSetup_WaiterAttrExport to inform NvSciStream they are done specifying waiter requirements.

```
NvSciError
NvSciStreamBlockElementWaiterAttrSet(
    NvSciStreamBlock const block,
    Uint32_t         const elemIndex,
    NvSciSyncAttrList const waitSyncAttrList
)
```

Specifying Requirements

Buffer data flows in only one direction, from the producer to the consumer(s). With synchronization objects, fences are generated at both endpoints and flow in both directions. Setting up synchronization resources begins by determining the requirements of each endpoint. The producer and consumers must query the NVIDIA drivers they will use with the appropriate APIs to obtain separate NvSciSyncAttrList handles. The handles represent the requirements for signaling (writing to) and waiting for (reading from) synchronization objects.

If an endpoint directly writes to or reads from the stream packets with the CPU instead of using an NVIDIA API, it does not generate fences and does not need synchronization signaling requirements. The endpoint must perform CPU waits for fences from the other endpoint and must create a waiting requirement attribute list with the NeedCpuAccess flag set.

In some cases, an application endpoint may require that packets it receives are available immediately, without waiting for a fence. This does not require synchronization waiting requirement attributes. Instead, the other endpoint is notified of CPU wait before sending the packets.

Once an endpoint determines its signal and wait requirements for synchronization objects, it stores the signal requirements locally and passes the wait requirements, one NvSciSyncAttrList per packet element, to the other endpoint through the stream. It passes an NvSciSyncAttrList with the wait requirement attribute, if fences are supported, into the endpoint block by calling NvSciStreamBlockElementWaiterAttrSet(). If the waiter for the element referenced by elemIndex does not support fences, NULL pointer is passed. The function call is the same for both producer and consumer endpoints. Ownership of the sync attribute handle remains with the caller. The stream creates a duplicate before the function returns.

After indicating the waiter requirements for all elements, the producer and consumer applications call the `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_WaiterAttrExport` to inform `NvSciStream` that specifying waiter requirements is complete.

```
NvSciError
NvSciStreamBlockElementWaiterAttrSet(
    NvSciStreamBlock const block,
    Uint32_t const elemIndex,
    NvSciSyncAttrList const waitSyncAttrList
)
```

Receiving Requirements

When the producer and consumer specify their synchronization requirements, the other endpoints receive a `NvSciStreamEventType_WaiterAttr` event. `NvSciStreamBlockElementWaiterAttrGet()` is operational on the endpoints receiving the event.

```
NvSciError
NvSciStreamBlockElementWaiterAttrGet(
    NvSciStreamBlock const block,
    Uint32_t const elemIndex,
    NvSciSyncAttrList* const waitSyncAttrList
)
```

If `NvSciStreamBlockElementWaiterAttrGet()` sets the output parameter to NULL, the element referenced by `elemIndex` synchronization objects cannot be used to coordinate with the other endpoint. The application must not allocate synchronization objects for the element. This case is rare, but producer and consumer application components designed to be fully modular must recognize and handle this situation.

Otherwise, the output parameter `waitSyncAttrList` is set to a synchronization attribute list handle. Ownership of this handle belongs to the application calling the function, and it must free the handle when it is no longer needed. The attributes in this list may not exactly match those specified by the originating endpoint. For more than one consumer, the stream combines their requirements into a single attribute list for the element. The stream itself may also perform transformations on the attributes to handle cross-process or cross-system cases.

The application must merge these wait requirements with its own signal requirements to form the final reconciled synchronization attribute list for the element. It can then use the final list to allocate a synchronization object from `NvSciSync`. A synchronization object is allocated per asynchronous element. The application must map these synchronization objects into the drivers and use them to generate fences passed into the stream.

After receiving all the sync attribute lists they care about, the endpoint applications call the `NvSciStreamBlockSetupStatusSet` function with a value of `NvSciStreamSetup_WaiterAttrImport`.

6.6.4.7.2.2 Synchronization Objects

Sending Objects

After allocating the synchronization objects, the application must inform the stream.

For each asynchronous element, it calls `NvSciStreamBlockElementSignalObjSet()`, which is operational after event `NvSciStreamEventType_WaiterAttr` is queried from the block. Ownership of the synchronization object handle remains with the caller. The stream creates a duplicate before the function returns.

After specifying all the synchronization objects, the endpoint applications call the `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_SignalObjExport`

```
NvSciError
NvSciStreamBlockElementSignalObjSet(
    NvSciStreamBlock const block,
    Uint32_t         const elemIndex,
    NvSciSyncObj    const SignalSyncObj
)
```

Receiving Objects

On the other endpoint receives the `NvSciStreamEventType_SignalObj`.

`NvSciStreamBlockElementSignalObjGet()` is operational on the endpoint receiving the event.

```
NvSciError
NvSciStreamBlockElementSignalObjGet(
    NvSciStreamBlock const block,
    Uint32_t         const queryBlockIndex,
    Uint32_t         const elemIndex,
    NvSciSyncObj*   const SignalSyncObj
)
```

Ownership of the retrieved `NvSciSyncObj` handle belongs to the application calling the function, and it must free the sync object when it is no longer required. The application must map these into the drivers and use them to interpret fences received from the stream.

As with the requirements, the synchronization objects received may not match those sent. If there is more than one consumer, the stream combines their synchronization objects into a single list per element before passing it to the producer. The stream may also replace the synchronization objects with its own if it must perform intermediate copy operations to pass the data from one endpoint to the other.

Synchronization specified for each element requires care. If a consumer requires synchronous access (Producer receives `NULL NvSciSyncObj` handle by `NvSciStreamBlockElementSignalObjGet()`) but the producer normally generates this data asynchronously, then an extra burden is placed on the producer application. During streaming the producer application waits for that element to finish being generated before it inserts the payload into the stream.

After receiving all the synchronization objects they care about, the endpoint applications call the `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_SignalObjImport`.

6.6.4.7.2.3 Comparison with EGL

When using EGLStreams, there is never any need to explicitly deal with synchronization objects. They are present in the NVIDIA EGL implementation, but require no action on the application's part. This is possible because the producer and consumer rendering libraries connect directly to an EGLStream and are able to coordinate synchronization setup themselves through the stream without the user being aware of it. In the NvSciStream model, the rendering libraries do not access the stream. Resources must be transferred between them and the stream. Therefore, these additional steps are required to initialize synchronization.

6.6.4.8 Frame Production

Once setup is complete, the producer application enters a cycle of receiving empty packets for reuse, writing to them, and inserting them back into the stream to be sent to the consumer(s).

When the pool indicates it exported all the packets, and the endpoints indicate they finished importing the packets and importing and exporting the sync objects, all blocks will receive a `NvSciStreamEventType_SetupComplete` event in the stream. This indicates that all necessary setup steps are complete. Applications that divide their event handling into separate initialization and runtime phases can use this to trigger the transition.



Note:

Resizing or otherwise replacing buffers during streaming is supported for non-safety builds and is described in a later section when it becomes available. It is not supported in the current release.

6.6.4.8.1 Obtaining Packets

When setup completes, the pool begins releasing the packets to the producer block for rendering. Subsequently, as payloads are returned from downstream because they were skipped or the consumer no longer needs them, they become available to the producer again. The order in which packets are received by the producer depends on various stream settings and is not deterministic, but the pool tries to optimize so that buffers with the least wait time until it is safe to write to them are available first.

When a packet becomes available for writing, a `NvSciStreamEventType_PacketReady` event is received by the producer block. The packet info is retrieved by a subsequent call. If multiple packets are available, the producer receives separate events for each of them.

After a packet becomes available, a producer application may call `NvSciStreamProducerPacketGet()` to obtain it. The cookie field is filled in with the cookie the producer assigned to the packet. With the packet handle, the producer application calls `NvSciStreamBlockPacketFenceGet()` to obtain the preferences for the packet elements. The preference field points to the location that will be filled with the

fence value indicating when the consumer indexed by queryBlockIndex is no longer using the data in the packet element's buffer.

```
NvSciError
NvSciStreamProducerPacketGet(
    NvSciStreamBlock const producer,
    NvSciStreamCookie *const cookie)

NvSciError
NvSciStreamBlockPacketFenceGet(
    NvSciStreamBlock const block,
    NvSciStreamPacket const handle,
    uint32_t const queryBlockIndex,
    uint32_t const elemIndex,
    NvSciSyncFence* const prefence
)
```

6.6.4.8.2 Writing Packets

Upon obtaining a packet, the producer application must ensure that the consumers are done reading from it before modifying their contents. If it is writing to the buffers using NVIDIA API, it must use the appropriate API-specific operation to insert a wait for each of the fences into the hardware command sequence. Then it looks up the API-specific buffer handle(s) for the packet and makes those buffers the current rendering targets. It may then proceed to issue rendering commands.

If instead the application writes directly to the buffer memory, it performs a CPU wait for the fences of the element buffers it wants to write to. Then it can begin writing the new data.

A producer application may retrieve multiple packets at once from the pool. It may operate on them at the same time and may insert the completed packets into the stream in any order, not necessarily in which they were retrieved.

6.6.4.8.3 Presenting Packets

When it finishes issuing its rendering instructions, the producer application must provide synchronization for their completion. For synchronous elements the producer must perform a CPU wait for rendering to finish on those synchronous elements before inserting the packet in the stream. For asynchronous elements, it instructs the APIs it used to generate fences to trigger when rendering finishes. The application sets the fences for the elements of the packet that is going to be presented by calling `NvSciStreamBlockPacketFenceSet()`.

The application can now insert the packet back into the stream along with the fences by calling `NvSciStreamProducerPacketPresent()` with the handle of the packet. The stream makes the packet available to consumers, performing any necessary copy or translations steps along the way to make the data and fences accessible. Once a packet is presented, the application must not attempt to modify its contents until it is returned for reuse.

```
NvSciError
```

```

NvSciStreamBlockPacketFenceSet(
    NvSciStreamBlock const block,
    NvSciStreamPacket const handle,
    uint32_t const elemIndex,
    NvSciSyncFence const* const postfence
)

NvSciError
NvSciStreamProducerPacketPresent(
    NvSciStreamBlock const producer,
    NvSciStreamPacket const handle
)

```

6.6.4.8.4 Comparison with EGL

With EGLStreams, the process for rendering and presenting frames varies depending on the rendering API chosen. For EGLSurface producers, the application never directly interacts with the individual image buffers. It simply issues rendering instructions for the surface and a swap command when it is done with each frame. The NvSciStream process therefore requires more hands-on interaction.

By contrast, use of CUDA and NvMedia producers for EGLStreams follows a very similar pattern to NvSciStream. After the first time a buffer is used, it is obtained from the stream when the consumer returns it. The application renders to the buffer, and then once again inserts it into the stream. The only real difference in buffer access is that with EGLStreams, the API's buffer handles are returned directly, whereas with NvSciStream the application receives a cookie and must look up the corresponding API handle.

For all EGLStream producers, a key difference in NvSciStream is the need to manage fences. In EGLStreams, the act of presenting a frame triggers automatic fence generation based on internal tracking of the last instructions issued for the buffer. Similarly, when buffers are returned, a fence from the consumer is internally associated with the buffer and the API waits for it the next time that buffer is used. In NvSciStream, the application must take control of requesting fences from the APIs to be inserted in the stream and waiting for fences received from the stream. This is necessary to support the more general variety of usage models that NvSciStream can handle that EGLStream cannot.

6.6.4.9 Frame Consumption

The consumer side cycle mirrors that of the producer, receiving packets full of data, reading from them, and returning them to the producer to be reused.

6.6.4.9.1 Acquiring Packets

Just as the producer block receives a NvSciStreamEventType_PacketReady event when a packet is available for reuse, the consumer block receives one when a packet containing new data arrives. If multiple packets are available, the consumer receives separate events for each of them.

After a packet becomes available, a consumer application may call `NvSciStreamConsumerPacketAcquire()` to obtain it. The cookie field is filled in with the cookie that the consumer assigned to the packet. With the packet handle, the consumer application calls `NvSciStreamBlockPacketFenceGet()` to obtain the preferences for the packet elements. The preference field points to the location that will be filled with the fence value indicating when the producer indexed by `queryBlockIndex` (always 0 for one producer) is no longer using the data in the packet element buffer.

Consumer packets are always received in the order that the producer sends them, but depending on the stream settings (such as if a mailbox queue is used) some packets may be skipped. The consumer may acquire and hold multiple packets at once.

```
NvSciError
NvSciStreamConsumerPacketAcquire(
    NvSciStreamBlock const consumer,
    NvSciStreamCookie *const cookie,
    NvSciSyncFence *const preferences
)
```

6.6.4.9.2 Reading Packets

Upon obtaining a packet, the consumer must wait until the contents are ready before reading from it. Any synchronous elements can be accessed right away. For all other elements, it must wait for the fences provided.

If it reads from the buffers using NVIDIA API, it must use the appropriate API-specific operation to insert a wait for each of the fences into the hardware command sequence. Then it can look up the API-specific buffer handles for the packet and make those buffers the current read sources. It can then issue commands that use the data.

If the application reads directly from the buffer memory, it must perform a CPU wait for all of the fences in the array. Then it can read the new data.

6.6.4.9.3 Releasing Packets

When it finishes issuing instructions to read from a packet, the consumer must return the packet to the producer for reuse. Packets may be returned in any order, regardless of the order they were acquired.

Before sending the frame back, the consumer provides synchronization to indicate when its operations complete. For synchronous elements, the consumer must perform a CPU wait for reading to finish before inserting the packet in the stream. For asynchronous elements, it must instruct the APIs it used to generate fences, which trigger when rendering finishes. The application sets the fences for the elements of the packet that is going to be released by calling `NvSciStreamBlockPacketFenceSet()`.

The application can now insert the packet back into the stream along with the fences by calling `NvSciStreamConsumerPacketRelease()` with the handle of the packet.

The stream returns the packet to the producer, performing any necessary copy or translations steps to make the fences accessible. When a packet is released, the

application must not attempt to read its contents until a new payload using the packet arrives.

```
NvSciError
NvSciStreamConsumerPacketRelease(
    NvSciStreamBlock const consumer,
    NvSciStreamPacket const handle,
    NvSciSyncFence const *const postfences
)
```

6.6.4.9.4 Comparison with EGL

As with the producer, consumer similarities between EGLStreams and NvSciStream depend on the rendering API chosen. For GL texture consumers, acquired buffers are bound directly to a selected texture, without the application knowing their details. CUDA and NvMedia consumers are closer to NvSciStream. The application receives individual buffers to read from and returns those to the stream when they are no longer required.

Again, the key difference in NvSciStream is the need to manage fences. Just as with the producer, EGLStreams handles all synchronization on behalf of the application, while with NvSciStream, the application is responsible for generating and waiting for fences.

6.6.4.10 Special Note Regarding Cache Coherence

The main application of NvSciStream is to produce data in one hardware engine and consume the data in another using different communication channels. Care must be taken in maintaining caching coherence for the buffers accessed by the engines. A memory-modifying operation made by one engine API may be outside of the knowledge of the engine API consuming the data. As a result, the engine consuming the data may be reading stale cached data. For example, memory-boundary IPC updates the GPU buffer without the knowledge of a CUDA consumer. Without proper cache invalidation, the CUDA consumer reading the GPU buffer using CUDA API may receive stale data. Consult with the documentation for the respective engine API for proper ways of invalidating the cache.

6.6.4.11 Disconnect Dead Consumer

In multicast streams, if a consumer is dead at runtime (either through abnormal termination or because it is stuck and unable to return the consumed packets), it can drain the packets in the pool and cause the entire stream to stop.

The application can monitor all consumers and disconnect the dead consumer process to continue streaming buffers to other consumers. After disconnecting the dead consumer, the stream can reuse the packets even if they are not released by the dead consumer. The fences generated by the dead consumer may never be expired. The application should wait on fences with a proper timeout value and skip waiting on fences from dead consumer.

```
NvSciError
NvSciStreamBlockDisconnect(
    NvSciStreamBlock const block
```

)

This API can only be used to disconnect the IpcSrc block connected to the dead consumer process after the stream setup complete.

6.6.4.12 Late-Attach/Re-Attach Consumer

In a multicast stream, late consumers can connect to the stream that is already in the streaming phase. Any consumer disconnected from the stream can also re-connect, which is treated as a new late-attach consumer.

Start Streaming with Early-connected Consumers

The stream can proceed to the streaming phase, even if not all consumers are connected, by calling `NvSciStreamBlockSetupStatusSet()` with a value of `NvSciStreamSetup_Connect` on the multicast block. This API can be called multiple times when a new consumer is connected to the multicast block.

The producer and early connected consumers perform the initialization steps as usual. To ensure all consumers can use the NvSciBuf/NvSciSync objects allocated by the producer, the producer applications must allocate these objects satisfying the constraints of both early-connected and late-attach consumers.

The producer can get the total number of supported consumers by calling `NvSciStreamBlockConsumerCountGet()`. The producer does not need to know which consumer is connected. It just queries the NvSciSync objects and fences from all consumers as usual. A NULL NvSciSync object or an empty fence is returned if the consumer is not connected. The producer can skip those NULL objects or empty fences.

Attach Late Consumers

Upon the completion of the stream initialization with all the connected consumers, the multicast block is ready to handle new connected consumers after querying the `NvSciStreamEventType_SetupComplete` event. The producer can call `NvSciStreamBlockSetupStatusSet()` with a value of `NvSciStreamSetup_Connect` again to start initialization with the late-attached consumers. The early-connected consumers can continue streaming without being affected or even aware of those late-attach consumers.

The late-attached consumers perform the same initialization steps as early connected consumers.

The producer only receives `NvSciStreamEventType_Signal10bj` event from the late-attach consumers. Because it does not know which consumer is newly connected, it needs to query the NvSciSync objects from all consumers. A NULL NvSciSync object is returned if it is not from the newly connected consumer.

- > If the retrieved sync object is NULL, then the producer can ignore it.
- > If the retrieved sync object is not NULL, then the producer should unregister the old sync object associated with this indexed consumer if any, and then register the new one.

After importing and mapping the sync objects from all late-attached consumers, the producer must call `NvSciStreamBlockSetupStatusSet()` with a value of `NvSciStreamSetup_SignalObjImport`.

Once the late-attach consumers complete all initialization steps, they can start streaming with the early-connected consumers.

Restrictions for Late-attach/Re-attach Consumer

- > Only one multicast block is allowed in the stream and must reside in the producer process.
- > No IpcSrc or IpcDst block can be connected between the producer block and the multicast block.
- > The new consumers can only be connected after all the existing consumers are in the streaming phase, not in the middle of initialization steps.
- > Late/re-attached consumers cannot send user-defined endpoint information to the producer.
- > If any late-attach consumer encounters an error or hangs during the initialization, all the late consumers that are involved in the initializations at that point must be disconnected and reconnected.

6.6.4.13 Teardown

In safety-certified builds, destruction of NvSciStream blocks and packets is not supported. Once created, all objects persist until the application shuts down. In non-safety builds, streams can be dynamically torn down at any time, and new streams can be created.

6.6.4.13.1 Packet Destruction

Packets can be destroyed one by one with the function:

```
NvSciError
NvSciStreamPoolPacketDelete(
    NvSciStreamBlock const pool,
    NvSciStreamPacket const handle
)
```

This API schedules an existing packet to be removed from the pool. If the packet is currently in the pool, it is removed right away. Otherwise, this is deferred until the packet returns to the pool. When the specified packet is returned to the pool, the pool releases resources associated with it and sends a `NvSciStreamEventType_PacketDelete` event to the producer and consumer. Once deleted, the packet may no longer be used for pool operations.

6.6.4.13.2 Block Destruction

Regardless of type, all blocks are destroyed with the same function:

```
NvSciError
NvSciStreamBlockDelete()
```

```
NvSciStreamBlock const block
)
```

When called, the block handle immediately becomes invalid, making it an error to use in any function call unless it is reused for a new block. Any blocks connected to this one are informed that the stream has disconnected, if they have not already, so that they may do an orderly cleanup. (It is not possible to connect a new block in place of a destroyed one.) Any resources associated with the block are scheduled for deletion, although this may not happen immediately if they are still in use by some part of the pipeline. If there is an NvSciEventNotifier bound to the block, it is unbound.

6.6.4.14 Safety Validation

This section applies to ASIL-D or ASIL-B(D) applications. To meet the ASIL-D and ASIL-B(D) safety goals when streaming data with NvSciStream, the application must perform extra validation steps. To ensure the data transmitted between producer and consumer applications are correct, NvSciStream provides a set of *WithCrc APIs to calculate the Cyclic Redundancy Checksum (CRC) values of the important initialization and runtime data and return the CRC values to the applications. The applications are responsible for verifying the CRC values before performing streaming and accessing the buffer data.

Use case 3 for the NvSciStream event sample illustrates the validation steps.

6.6.4.14.1 Stream Setup

During the stream setup, the application should call the new set of *WithCrc APIs to get and save the CRC values. Before entering the streaming phase, the producer application must verify the CRC values and other validation data provided by all ASIL-D consumers and indicate whether it is safe to start streaming.

The producer application must maintain the init-time CRC values of:

- > Producer's user-defined endpoint information
- > Consumer's user-defined endpoint information (optional)
- > The endpoint information from each consumer may be different; the producer must maintain an array of CRC values for each consumer
- > Buffer information
 - The consumer will not receive another consumer's CRC buffer object, so the producer must maintain an array of buffer CRC values for each consumer
- > Producer's synchronization information
- > Consumer's synchronization information
 - The consumer may provide different synchronization information, so the producer needs to maintain an array of synchronization CRC values for each consumer

The consumer application needs to maintain the CRC init-time values of:

- > Producer's user-defined endpoint information
- > Consumer's user-defined endpoint information (optional)
- > Buffer information

- > Producer's synchronization information
- > Consumer's synchronization information

Create Blocks

The ASIL-D or ASIL-B(D) producer and consumers must indicate whether they require CRC validation when creating the blocks by calling `NvSciStreamProducerCreate2()` with `crcValidate` set to true. Similarly, the consumer application can call `NvSciStreamConsumerCreate2()` with `crcValidate` set to true.

User-defined Endpoint Information

When the producer/consumer provides the endpoint information to the other side, it calls `NvSciStreamBlockUserInfoSetWithCrc()` to send the endpoint information and calls `NvSciStreamBlockUserInfoGetWithCrc()` to query the endpoint information.

The producer application must define a unique stream ID for each stream and send it to all consumer applications as part of the endpoint information. The consumers share this stream ID along with the CRC values the producer to validate if the producer process and consumer process are connected to the same stream.

The application must query the user information by different `userType` in the same order as that set by the application at remote ends. As CRC is generated based on the order of information set or retrieved, the order of setting the user information and retrieving the user information must be same on both ends.

Query Connection Index

Once the consumer receives the `NvSciStreamEventType_Connected` event, The consumer calls `NvSciStreamConsumerIndexGet()` to query the connection index. It must share the connection index with the producer along with the `NvSciLpc` endpoint name in a later validation step. The producer application can tell which indexed consumer is ASIL-D or ASIL-B(D) according to the `NvSciLpc` channel.

Packet Layout

The application must set up buffer attributes to allocate extra CRC buffers in each packet for each endpoint, because they need to exchange the CRC values and additional validation data in the shared buffer instead of the regular communication channel used for sending and receiving messages. For example, in a stream with two consumers, it needs to allocate three CRC buffers per packet: one for the producer and the remaining two for each consumer.

When the pool sends the final layout, it calls `NvSciStreamBlockElementAttrSetWithCrc()` to set the reconciled element attributes. All CRC buffer objects are visible to the producer reapplication, but the consumer application should only access the producer's CRC element and its own CRC element. The pool calls `NvSciStreamBlockElementNotVisible()` to mark the consumer's CRC buffer objects only visible to the owner, who has read-write permission.

The producer/consumer applications must call `NvSciStreamBlockElementAttrGetWithCrc()` to query the reconciled attributes for each

element in the same order in which the pool set the attributes, such as from element index 0 to element index (element_count - 1).

Buffer Objects

The pool creates new packets one by one. The next new packet is created after receiving the packet acceptance status from producer and consumers, and it calls `NvSciStreamPoolPacketInsertBufferWithCrc()` to add the buffer objects to the new packet.

The producer/consumer applications call `NvSciStreamBlockPacketBufferGetWithCrc()` to query the buffer objects from each packet. The producer application must track CRC values for each consumer by providing an array of CRC values to this call. According to the element type and the index of the consumer, the producer can tell if the indexed element is a CRC element for the indexed consumer, and it only updates the CRC value for that consumer.

Synchronization Objects

The producer/consumer calls `NvSciStreamBlockElementSignalObjSetWithCrc()`/
`NvSciStreamBlockElementSignalObjGetWithCrc()` to set/query the synchronization objects. The applications must set or get the information in the same order, such as from element index 0 to element index (element_count - 1).

When the producer queries the synchronization objects from each consumer, it should maintain the CRC values for different consumers.

Validation Event

When the consumer application completes the buffer and synchronization setup steps, it writes the validation data into its own CRC element of the first created packet and calls `NvSciStreamBlockSetupStatusSet()` with `NvSciStreamSetup_CrcImport` to inform the producer that the initialization-time validation data is ready. The validation data should include the following:

- A user-defined magic ID to indicate it is init-time validation data
- All the CRC init-time values maintained by the consumer
- The stream ID received from the producer as one part of the endpoint information
- A connection index of the consumer
- The NvScilpc endpoint name used by the consumer process

When the producer application receives the `NvSciStreamEventType_Validate` event, the producer application must verify the data provided by each ASIL-D or ASIL-B(D) consumer in the consumer's CRC elements of the first created packet:

- The magic number matches the init-time magic number
- The stream ID is the same as that sent by the producer
- The consumer index is unique and in a valid range
- Determine which indexed consumer is the ASIL-D or ASIL-B(D) consumer according to the NvScilpc endpoint name provided by the consumer. The Producer application must keep track of the consumer indexes of ASIL-D or ASIL-B(D) consumers and must validate the data only from those consumers

- > The CRC values provided by the consumers are the same as those calculated by the producer

The producer application also must validate the received buffer and synchronization objects to meet its buffer/ synchronization requirement.

- > Retrieve the reconciled attribute list from the buffer/synchronization objects via `NvSciBufObjGetAttrList`/ `NvSciSyncObjGetAttrList`
- > Validate that the reconciled attribute list meets its own requirements via `NvSciBufAttrListValidateReconciledAgainstAttrs`/ `NvSciSyncAttrListValidateReconciledAgainstAttrs`
- > Validate the buffer/synchronization objects satisfies the constraints of the the associated reconciled attribute list via `NvSciBufObjValidate`/ `NvSciSyncObjValidate`

The producer application must query the max buffer access permission for the lower ASIL consumers on the reconciled attribute list associated with the `NvSciBufObj`(s) using `NvSciBufObjGetMaxPerm`.

Only when the validation process is complete, the producer application can inform the stream to start streaming by calling `NvSciStreamBlockSetupStatusSet()` with `NvSciStreamSetup_CrcImport`.

6.6.4.14.2 Streaming

During streaming, the applications must validate the correctness of the payload before accessing the buffer data.

Frame Production

After the producer application receives a new packet, it must validate that the received cookie is one of those provided to the `NvSciStream` during packet creation and perform the following validation steps:

- > Clear the magic number in the producer's CRC element, in case that the consumer tries to access this packet.
- > For each consumer:
 - Verify that the magic number in the consumer's CRC buffer matches the user-defined runtime magic number
 - If the magic number matches, but CRC value is 0, the packet may be returned from the mailbox queue. You can skip validating the CRC value. Otherwise, reset the input CRC value to 0 and pass it to `NvSciStreamBlockPacketFenceGetWithCrc()` when querying the pre-fences from the consumer. Verify that it is the same as the CRC value in the consumer's CRC buffer. You can skip validating the CRC values if the packet is in use first and not released from consumers
 - Clear the data in the consumer's CRC element and fill in the runtime magic number
- > Write the data into the data element once validation is complete

- > Before presenting the packet to the consumers, the producer application must write the validation data into producer's CRC element:
 - Reset the input CRC to 0 and pass it to `NvSciStreamBlockPacketFenceSetWithCrc()` when setting all the post-fences to get the updated fence CRC value
 - Write the runtime magic number, fence CRC, frame count, and current timestamp to the producer's CRC element.

Frame Consumption

After the consumer application acquires a new packet, it must validate that the received cookie is one of those provided to the NvSciStream during packet creation and perform following validation steps:

- > Clear the magic number in its consumer's CRC element in case the producer tries to access this packet
- > Reset the input CRC value to 0 and pass it to `NvSciStreamBlockPacketFenceGetWithCrc()` when querying all of the pre-fences from the producer
- > Perform the following validation steps before accessing buffer data:
 - Verify that the magic number in the producer's CRC buffer matches the user-defined runtime magic number
 - Reset the input CRC value to 0 and pass it to `NvSciStreamBlockPacketFenceGetWithCrc()` when querying all the pre-fences from the producer. Verify that it is the same value as the CRC value in the producer's CRC buffer
 - The frame count should be incremented. With FIFO queue, it should be incremented by 1 for no packet drops.
 - With a mailbox queue, if the application only wants to process the latest data, it can check the timestamp of when the producer sends the data to decide if it is acceptable.
- > Read the data from the data element once validation is complete.
- > Before releasing the packet back to the producer for reuse, the consumer application must write the validation data into producer's CRC element:
 - Reset the input CRC to 0 and pass it to `NvSciStreamBlockPacketFenceSetWithCrc()` when setting all the post-fences to get the updated fence CRC value.
 - Write the runtime magic number and fence CRC into its consumer's CRC element.

6.6.4.14.3 Stream with Late-Attach/Re-Attach Consumers

- > Late-attach ASIL-D or ASIL-B(D) consumers should perform the same validation steps.
- > The producer application should track which indexed ASIL-D or ASIL-B(D) consumers are in the streaming phase and only validate the CRC values from these active ASIL-D or ASIL-B(D) consumers before accessing the buffer data.

- > If the producer application calls `NvSciStreamBlockDisconnect()` to disconnect the consumer branch, it should skip validating the data from this consumer's CRC element.
- > Before the consumer application calls `NvSciStreamBlockDelete()` to disconnect, it must clear the data in the consumer CRC element and add the magic number into the CRC element of all packets held by this consumer.

6.6.4.14.4 Restrictions for ASIL-D or ASIL-B(D) CRC Validation

Following are restrictions for the ASIL-D or ASIL-B(D) CRC validation:

- > If application is using multicast block in the stream, all ASIL-D/ASIL-B(D) consumers must connect directly to one multicast block, which must reside in the producer process and have no other multicast block before it.
- > The producer and consumer applications should set and query the information in a specific element index order or element type order.
- > The unused element feature is not supported with ASIL-D or ASIL-B(D) consumers.
- > PresentSync block is not supported with ASIL-D or ASIL-B(D) producer or consumers.
- > ReturnSync block is not supported with ASIL-D or ASIL-B(D) consumers.

6.6.4.15 NvSciStream Sample Application

The NVIDIA SDK provides a single sample application to demonstrate how to use the NvSciStream API to build simple and complex streams. This application combines all the features of the multiple separate samples provided in previous versions of the SDK and illustrates some new ones. This includes:

- > Both single- and multi-cast streaming
- > Both intra-process, inter-process, and inter-chip streaming, or a combination of the three when multicasting
- > Waiting for events on single blocks using the NvSciStream functions directly or on all blocks at once using an NvSciEventService
- > CUDA to CUDA streaming and NvMedia to CUDA streaming

Migration

Those familiar with previous samples should be aware of several changes:

The complex C++ classes are eliminated, using flatter C code instead. The top-level setup to connect all the NvSciStream blocks can be found in the `main.c` file, while all other block operations are split into separate files for each block type. This provides a clearer example of the required function calls for each kind of block.

Instead of performing a specific sequence of operations and waiting for specific events to arrive at each block, this application supports a general event loop driven model. The recommended approach is to use NvSciEventNotifiers generated for each block, and a single main thread which can wait for events on all blocks simultaneously. When an event arrives on a block, it is directed to an appropriate block-specific function to handle it. Events not associated with NvSciStream can also be bound to NvSciEventNotifiers and be handled in the same loop. This makes for a more robust application design.

Those preferring to handle each NvSciStream block separately can still wait for events on individual blocks. The sample illustrates this approach as well.

Prerequisites

NvScilpc

With inter-process streaming, the sample applications stream packets between a producer process and a consumer process via inter-process communication (NvScilpc) channels.

The NvScilpc channels are configured via a plain text file, /etc/nvsciipc.cfg. For more information, see [NvScilpc Configuration Data](#). The recommended NvScilpc channels for these sample applications are as follows:

INTER_PROCESS	nvscistream_0	nvscistream_1	16	24576
INTER_PROCESS	nvscistream_2	nvscistream_3	16	24576
INTER_PROCESS	nvscistream_4	nvscistream_5	16	24576
INTER_PROCESS	nvscistream_6	nvscistream_7	16	24576

Where inter-chip streaming is used, the sample application streams packets between different chips via NvScilpc (INTER_CHIP, PCIe) channels. For more information, see [Chip to Chip Communication](#).

CUDA

This sample application uses the CUDA toolkit. Ensure CUDA toolkit is installed. See [Installing CUDA Debian Packages](#).

Building the NvSciStream Event-Driven Sample Application

The NvSciStream sample includes source code and a Makefile.

1. On the host system, navigate to the sample application directory:

```
cd <top>/drive-linux/samples/nvsci/nvscistream/event/
```
2. Build the sample application:

```
make clean  
make
```

Running the NvSciStream Event-Driven Sample Application

By default, the event-driven sample application will create a single-process unicast stream from CUDA to CUDA, using a mailbox queue for the consumer, and handling all events in a single loop.

This behavior can be modified with the following command line switches. If running in multiple processes, “-p” and “-c” must be specified for the producer and all consumers or the stream will not fully connect. Producer and consumers do not all need to reside in separate processes and can be combined. Some options only affect the setup of the producer or one of the consumers and are ignored if specified in the wrong process.

Option	Meaning	Default
-m <count>	Specifies the number of consumers. Set in the producer process.	1
-f <count>	Specifies the number of packets to create. Set in the producer process.	3
-l <index> <limit>	Add a limiter block between the producer and the indexed consumer, with the specified packet limit. Set in the producer process.	
-q <index> {f m}	Use a fifo (f) or mailbox (m) queue for the indexed consumer. Set in the consumer process.	f
-e {s t}	Handle events through a single event service (s) or through separate per-thread event loops for each block (t). Set in each process.	s
-E	Use the user-provided event service to handle internal I/O messages on ipc blocks. Only supported with event service.	
-s {y r}	Image format. y : NvSciColor_Y8U8Y8V8 Image Color Format in use case 2 r : NvSciColor_A8R8G8B8 Image Color Format in use case 2	r

Option	Meaning	Default
-u <index>	<p>Use case index:</p> <p>1: CUDA to CUDA</p> <p>2: NvMedia to CUDA</p> <p>3: CUDA to CUDA with safety validation. Not supported in C2C</p> <p>Must be set the same in all processes.</p>	1
-i	<p>Optional. Set endpoint info by producer and consumers in this process and query info from other endpoints.</p> <p>Set in each process.</p>	
For inter-process operation:		
-p	Producer resides in this process	
-c <index>	Indexed consumer resides in this process	
For inter-chip operation:		
-P <index> <ipc endpoint>	Producer resides in this process. NvScilpc endpoint used by this producer to communicate with the indexed consumer in another chip.	

Option	Meaning	Default
-C <index> <ipc endpoint>	Indexed consumer resides in this process but in a different chip from the producer, and the NvScilpc endpoint used by this consumer. “-C” and “-c” can't be used simultaneously in one process.	
-F <index> <count>	Specify the number of packets in the pool attached to the memory boundary IpcDst block of indexed C2C consumer. Set in the consumer process.	3
-Q <index> {f m}	Specify a fifo (f) or mailbox (m) queue attached to the memory boundary IpcSrc block associated with the indexed consumer. Set in the producer process.	

1. Copy the sample application to the target filesystem:

```
cp <top>/drive-linux/samples/nvsci/nvscistream/event/nvscistream_event_sample
      <top>/drive-linux/filesystem/targetfs/home/nvidia/
```

2. Following are several examples of how to run the sample application with different configurations:

- > Single-process unicast with default setup:

```
./nvscistream_event_sample
```

- > Single-process with three consumer multicast, NvMedia to CUDA streaming, and per-block event threads:

```
./nvscistream_event_sample -m 3 -u 2 -e t
```

- > Two consumers, with one in the same processes as the producer and the other in a separate process. Both enable the endpoint info option:

```
./nvscistream_event_sample -m 2 -p -c 0 -i &
```

```
./nvscistream_event_sample -c 1 -i &
```

- > Three consumers, with one in the same process as the producer and two in a separate process, and a mailbox queue for one of them:

```
./nvscistream_event_sample -m 3 -p -c 0 &
```

```
./nvscistream_event_sample -c 1 -c 2 -q 2 m &
```

- > Multi-process cuda/cuda stream with one consumer on another SoC. A FIFO queue is attached to the memory boundary IpcSrc block, and a 3-packet pool is attached to the memory boundary IpcDst block. It uses the NvScilpc channel <nvscic2c_pcnie_s0_c5_1> <nvscic2c_pcnie_s0_c6_1>.

On chip s0:

```
./nvscistream_event_sample -P 0 nvscic2c_pcnie_s0_c5_1 -Q 0 f
```

On chip s1:

```
./nvscistream_event_sample -C 0 nvscic2c_pcnie_s0_c6_1 -F 0 3
```

- > Four consumers, with one in the same process as the producer, one in another process but on the same chip as the producer, and two in another process on another chip.

Both the 3rd and 4th consumers have mailbox queue attached to the memory boundary IpcSrc block and 5-packet pool attached to the memory boundary IpcDst block.

Inter-chip NvScilpc channels used by the 3rd and 4th consumers:

- <nvscic2c_pcnie_s0_c5_1> <nvscic2c_pcnie_s0_c6_1>
- <nvscic2c_pcnie_s0_c5_2> <nvscic2c_pcnie_s0_c6_2>

On chip s0:

```
./nvscistream_event_sample -m 4 -c 0 -q 0 m -Q 2 m -Q 3 m -P 2
nvscic2c_pcnie_s0_c5_1 -P 3 nvscic2c_pcnie_s0_c5_2 &
./nvscistream_event_sample -c 1 -q 1 m &
```

On chip s1:

```
./nvscistream_event_sample -C 2 nvscic2c_pcnie_s0_c6_1 -q 2 f -F 2 5 -C 3
nvscic2c_pcnie_s0_c6_2 -q 3 m -F 3 5
```



Note:

The nvscistream_event_sample application must be run as root user (with sudo).

If the nvscistream_event_sample application fails to open the IPC channel, cleaning up NvScilpc resources may help.

```
sudo rm -rf /dev/mqueue/*
sudo rm -rf /dev/shm/*
```

For inter-chip use cases:

Ensure different SoCs are set with different Soc IDs. See the "Bind Options for SOC ID for C2C in GOS-DT" section in the NVIDIA DRIVE OS Linux PDK Developer Guide.

6.6.4.16 NvSciStream Performance Test Application

NvSciStream provides a test application to measure KPIs when streaming buffers between a CPU producer and CPU consumers. This test focuses on NvSciStream performance, which does not use CUDA, NvMedia, or other hardware engines. To simplify measuring packet-delivery latency for each payload, the stream uses FIFO mode.

This test application is for performance testing purposes. It may simplify some setup steps and set unnecessary synchronization objects or fences to the CPU endpoints to include the fence transport latency in the measurement. To see how to create a stream with NvSciStream API, refer to [NvSciStream Sample Application](#)

This test uses the NvPlayFair library (see [Benchmarking Library](#)) to record timestamps, set the rate limit, save raw latency data, and calculate the latency statistics (such as the min, max, and mean value) on different platforms and operating systems.

The test app supports a variety of test cases:

- > Single-process, inter-process, and inter-chip streaming
- > Unicast and multicast streaming

The test can set different stream configurations:

- > Number of packets allocated in pool.
- > Number of payloads transmitted between producer and consumers.
- > Buffer size for each element.
- > Number of synchronization objects used by each endpoint.
- > Frame rate, frequency of the payloads presented by the producer.
- > Memory type, vidmem, or sysmem.

The test measures several performance KPIs:

- > Latency for each process:
 - Total initialization time
 - Stream setup time
 - Streaming time
- > Latency for each payload:
 - Duration to wait for an available or ready packet
 - End-to-end packet-delivery latency
- > PCIe bandwidth in inter-chip stream

The README file in the test folder explains these KPIs with more details.

Prerequisites

NvScilpc

Where inter-process streaming is used, the performance test application streams packets between a producer process and a consumer process using inter-process communication (NvScilpc) channels.

The NvScilpc channels are configured using `/etc/nvscilpc.cfg`, on Linux. For more information on NvScilpc configuration data, see [NvScilpc Configuration Data](#). The NvScilpc channels used by the performance test application are as follows:

```
INTER_PROCESS nvscistream_0 nvscistream_1    16 24576
INTER_PROCESS nvscistream_2 nvscistream_3    16 24576
INTER_PROCESS nvscistream_4 nvscistream_5    16 24576
INTER_PROCESS nvscistream_6 nvscistream_7    16 24576
```

Where inter-chip streaming is used, the sample application stream packets between different chips via NvScilpc (INTER_CHIP, PCIe) Channels. For more information, see [Chip to Chip Communication](#).

NvPlayFair

This performance test application uses the performance utility functions in the [NvPlayFair library](#).

Time Sync

For inter-chip use cases, accurate packet-delivery latency requires synchronizing time on two SoCs using PTP. For more information, refer to "AVNU PTP for Development" in [Orin Time Sync](#). The process uses NvPPS APIs to get the synced PTP time.

Building the NvSciStream Performance Test Application

The NvSciStream performance test includes source code, README, and a Makefile.

On the host system, navigate to the test directory:

```
cd <top>/drive-linux/samples/nvsci/nvscistream/perf_tests/
```

Build the performance test application:

```
make clean  
make
```

Running the NvSciStream Performance Test Application

Option	Meaning	Default
-h	Prints supported test options	
-n <count>	Specifies the number of consumers. Set in the producer process.	1
-k <count>	Specifies the number of packets in pool. Set in the producer process for primary pool. Set in the consumer process for c2c pool.	1
-f <count>	Specifies the number of payloads. Set in all processes.	100

Option	Meaning	Default
-b <size>	Specifies the buffer size (MB) per packet.	1
-s <count>	Specifies the number of sync objects per client. Set by each process.	1
-r <count>	Specifies the producer frame-present rate (fps)	
-t <0 1>	Specifies the memory type. 0 for sysmem and 1 for vidmem. Pass dGPU UUID using -u, if using vidmem.	0
-u	Required for vidmem buffers. Can be retrieved from 'nvidia-smi -L' command on x86	
-l	Measure latency. Skip if vidmem is used. Set in all processes.	False
-v	Save the latency raw data in csv file. Ignored if not measuring latency.	False
-a <target>	Specifies the average KPI target (us) for packet-delivery latency. Compare the test result with the input target with 5% tolerance. Ignored if not measuring latency.	
-m <target>	Specifies the 99.99 percentile KPI target (us) for packet-delivery latency. Compare the test result with the input target with 5% tolerance. Ignored if not measuring latency.	
For inter-process operation:		

Option	Meaning	Default
-p	Inter-process producer.	
-c <index>	Inter-process indexed consumer.	
For inter-chip operations:		
-P <index> <ipc endpoint>	Inter-SoC producer, NvScilpc endpoint name connected to indexed consumer.	
-C <index> <ipc endpoint>	Inter-SoC consumer, NvScilpc endpoint used by this indexed consumer.	

Copy the sample application to the target filesystem:

```
cp <top>/drive-
linux/samples/nvsci/nvscistream/perf_tests/test_nvscistream_perf
<top>/drive-linux/targetfs/home/nvidia/
```

Following are examples of running the performance test application with different configurations:

- > Measure latency for single-process unicast stream with default setup:

```
./test_nvscistream_perf -l
```

- > Measure latency for single-process unicast stream with three packets in pool:

```
./test_nvscistream_perf -l -k 3
```

- > Measure latency for single-process multicast stream with two consumers:

```
./test_nvscistream_perf -n 2 -l
```

- > Measure latency for inter-process unicast stream with default setup:

```
./test_nvscistream_perf -p -l &
```

```
./test_nvscistream_perf -c 0 -l
```

- > Measure latency for inter-process unicast stream with a fixed producer-present rate at 100 fps, which transmits 10,000 payloads:

```
./test_nvscistream_perf -p -f 10000 -l -r 100 &
```

```
./test_nvscistream_perf -c 0 -f 10000 -l
```

- > Measure latency and save raw latency data in nvscistream_*.csv file for inter-process unicast stream, which transmits 10 payloads:

```
./test_nvscistream_perf -p -f 10 -l -v &
```

```
./test_nvscistream_perf -c 0 -f 10 -l -v
```

- > Measure PCIe bandwidth for the inter-chip unicast stream with 12.5 MB buffer size per packet, which transmits 10,000 frames. The two commands are run on different SoCs with <pcie_s0_1> <pcie_s1_1> PCIe channel:

On chip s0:

```
./test_nvscistream_perf -P 0 pcie_s0_1 -l -b 12.5 -f 10000
```

On chip s1:

```
./test_nvscistream_perf -C 0 pcie_s1_1 -l -b 12.5 -f 10000
```



Note:

The test_nvscistream_perf application must run as root user (with sudo).

For the inter-process use case:

If it fails to open the IPC channel, cleaning up NvScilpc resources may help.

```
sudo rm -rf /dev/mqueue/*
sudo rm -rf /dev/shm/*
```



Note: For inter-chip use cases:

Ensure different SoCs are set with different SoC IDs. For Tegra-x86 use cases, set a non-zero SoC ID on the NVIDIA Tegra side, because x86 uses 0 as the SoC ID. For more information, refer to the "Bind Options" section in the AV PCT Configuration chapter of the *NVIDIA DRIVE OS 6.0 Developer Guide*.

6.6.5 Synchronization

This section describes how to set up synchronization objects required by an application or a set of applications, and how to use them to control the order of operations by NVIDIA hardware. The basic setup process is similar to the process used for allocating buffers described in the previous chapter:

- > Specify the restrictions imposed by the hardware components that signal the sync objects.
- > Specify the restrictions imposed by the hardware components that wait on the sync fences.
- > Gather the information for each set of sync objects into one application, which allocates them.
- > Share the allocated sync objects with other applications.
- > Map the sync objects into UMD-specific interfaces.

The streaming process issues commands to update the sync objects, and commands to wait for those updates, so that different sets of operations remain in sync with each other.

6.6.5.1 Terminology

Agent: An entity in the system that executes instructions. An agent may be a CPU thread and also a hardware engine. An agent is the entity that interacts with actual hardware primitives abstracted by NvSciSync. The goal of NvSciSync is to synchronize agents.

6.6.5.2 Synchronization Basics

In an NVIDIA hardware system, there are typically multiple execution agents running simultaneously. For example: CPU, GPU, and other engines. Task interfaces to engines behave asynchronously. The CPU prepares a job for it and queues it in the interface. There might be multiple jobs pending on an engine and they are scheduled automatically in the hardware. CPU and applications don't know upfront the order in which jobs are executed.

This creates a need for job synchronization with dependencies between them, like a pipeline of several engines working on the same data, which preferably should not involve expensive CPU intervention.

NVIDIA hardware supports multiple synchronization mechanisms that solve this problem. They are context sensitive and not every engine understands all the mechanisms.

NvSciSync provides an abstraction layer that hides details of synchronization primitives used in a concrete situation. One of the most basic concepts of NvSciSync is a sync object. It abstracts a single instance of a specific synchronization primitive. A sync object has a current state and can be signaled. Signaling a sync object moves it to the next state. Normally, an application developer associates a sync object with a chain of events that must occur in the same order. For example, a video input engine may always signal the same sync object after producing a camera frame. This way, the sync object can be inspected at any time to check which frames were already written. A sync object must only be signaled by a single agent. An agent that signals (at the request of an application) is called a signaller.

Another basic concept of NvSciSync is a sync fence. A sync fence is associated with a specific sync object and contains a snapshot of that object's state. A fence is considered expired if its snapshot is behind or equal to the current state of the object. A fence whose state has not yet been reached by the object is said to be pending. Usually, multiple fences are associated with a single sync object and might correspond to different states of that object. A sync fence is generated by the signaller application and shared with others. An application can make an agent wait on a fence. An agent waiting on a sync fence is called a waiter in the context of the given sync object.

6.6.5.3 NvSciSync Module

To use NvSciSync you must first open an NvSciSyncModule. This module represents the library's instance created for that application and acts as a container for other NvSciSync resources. Typically, there is only a single NvSciSyncModule in an application but having all resources contained in NvSciSyncModule allows multiple threads or other libraries to use NvSciSync in an isolated manner. All other NvSciSync resources are associated with an NvSciSyncModule on creation.

6.6.5.3.1 NvSciSyncModule

```
NvSciSyncModule module = NULL;
NvSciError err;
err = NvSciSyncModuleOpen(&module);
if (err != NvSciError_Success) {
    goto fail;
}
/* ... */
NvSciSyncModuleClose(module);
```

6.6.5.4 Inter-Application

If there are multiple processes involved, all communication of NvSciSync structures should go via NvScilpc channels. Each application needs to open its own Ipc endpoints.

6.6.5.4.1 NvScilpc init

```
NvSciIpcEndpoint ipcEndpoint = 0;
NvSciError err;
err = NvSciIpcInit();
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciIpcOpenEndpoint("ipc_endpoint", &ipcEndpoint);
if (err != NvSciError_Success) {
    goto fail;
}
/* ... */
NvSciIpcCloseEndpoint(ipcEndpoint);
NvSciIpcDeinit();
```

6.6.5.5 NvSciSync Attributes

NvSciSync clients must supply the properties and constraints of an NvSciSync object to NvSciSync before allocating the object. This is expressed with attributes. An attribute is a key - value pair. You can view all supported keys in the header files together with value types that can be used with them.

Each application wanting to use a sync object indicates its needs in the form of various attributes before the sync object is created. Those attributes are then communicated to the signaler, who gathers all applications' attributes and has NvSciSync reconcile them. Successful reconciliation creates a new attribute list satisfying all applications constraints. The signaler then allocates a sync object using resources described by those attributes. This sync object, together with reconciled attributes list, is then shared with all waiters that need access to this sync object.

6.6.5.5.1 NvSciSync Attributes List

Attributes coming from a single source are kept in an attribute list structure.

6.6.5.5.1.1 NvSciSyncAttrList

```
NvSciSyncAttrList attrList = NULL;
NvSciError err;
/* create a new, empty attribute list */
err = NvSciSyncAttrListCreate(module, &signalerAttrList);
if (err != NvSciError_Success) {
    goto fail;
}
/*
 * fill the list - this example corresponds to a CPU signaller
 * that only needs to signal but not wait
*/
NvSciSyncAttrKeyValuePair keyValue[2] = {0};
bool cpuSignaler = true;
keyValue[0].attrKey = NvSciSyncAttrKey_NeedCpuAccess;
keyValue[0].value = (void*) &cpuSignaler;
keyValue[0].len = sizeof(cpuSignaler);
NvSciSyncAccessPerm cpuPerm = NvSciSyncAccessPerm_SignalOnly;
keyValue[1].attrKey = NvSciSyncAttrKey_RequiredPerm;
keyValue[1].value = (void*) &cpuPerm;
keyValue[1].len = sizeof(cpuPerm);
err = NvSciSyncAttrListSetAttrs(list, keyValue, 2);
if (err != NvSciError_Success) {
    goto fail;
}
/* ... */
NvSciSyncAttrListFree(signalerAttrList);
```

6.6.5.5.1.2 Reconciliation

After gathering all attribute lists, the signaller application must reconcile them. Successful reconciliation results in a new, reconciled attribute list that satisfies all applications' requirements. If NvSciSync cannot create such a list because attributes contradict, it instead creates an attribute list that describes the conflicts in more detail. In the example below, assume that waiterAttrList1 and waiterAttrList2 were created in the same process, so the variables are visible.

```
NvSciSyncAttrList unreconciledList[3] = {NULL};
NvSciSyncAttrList reconciledList = NULL;
NvSciSyncAttrList newConflictList = NULL;
NvSciError err;
unreconciledList[0] = signalerAttrList;
unreconciledList[1] = waiterAttrList1;
unreconciledList[2] = waiterAttrList2;
err = NvSciSyncAttrListReconcile(
    unreconciledList,           /* array of unreconciled lists */
    3,                         /* size of this array */
    &reconciledList,           /* output reconciled list */
    &newConflictList);        /* conflict description filled in case of
                                reconciliation failure */
if (err != NvSciError_Success) {
    goto fail;
}
/* ... */
```

```
NvSciSyncAttrListFree(reconciledList);
NvSciSyncAttrListFree(newConflictList);
```

NvSciSync recognizes which attribute lists are reconciled and which are not. Some NvSciSync APIs that take NvSciSyncAttrList expect the list to be reconciled.

6.6.5.5.2 Inter-Application

The above example assumes that waiterAttrList1 and waiterAttrList2 were received from the waiter applications. Hence, no cross-process semantics were required. If a waiter lives in another process, then the attributeList must be exported to a descriptor first, communicated via NvScilpc, and then imported on the receiver's end. In some cases, there are multiple lists traveling through multiple NvScilpc channels.

6.6.5.5.2.1 Export/Import NvSciSyncattrList

```
/* waiter */
NvSciSyncAttrList waiterAttrList = NULL;
void* waiterListDesc = NULL;
size_t waiterListDescSize = 0U;
NvSciError err;
/* creation of the attribute list, receiving other lists from other waiters */
err = NvSciSyncAttrListIpcExportUnreconciled(
    &waiterAttrList,           /* array of unreconciled lists to be
exported */
    1,                         /* size of the array */
    ipcEndpoint,               /* valid and opened NvSciIpcEndpoint
intended to send the descriptor through */
    &waiterListDesc,          /* The descriptor buffer to be allocated
and filled in */
    &waiterListDescSize );     /* size of the newly created buffer */
if (err != NvSciError_Success) {
    goto fail;
}
/* send the descriptor to the signaler */
NvSciSyncAttrListFreeDesc(waiterListDesc);
/* signaller */
void* waiterListDesc = NULL;
size_t waiterListDescSize = 0U;
NvSciSyncAttrList unreconciledList[2] = {NULL};
NvSciSyncAttrList reconciledList = NULL;
NvSciSyncAttrList newConflictList = NULL;
NvSciSyncAttrList signallerAttrList = NULL;
NvSciSyncAttrList importedUnreconciledAttrList = NULL;
/* create the local signallerAttrList */
/* receive the descriptor from the waiter */
err = NvSciSyncAttrListIpcImportUnreconciled(module, ipcEndpoint,
    waiterListDesc, waiterListDescSize,
    &importedUnreconciledAttrList);
if (err != NvSciError_Success) {
    goto fail;
}
/* gather all the lists into an array and reconcile */
unreconciledList[0] = signallerAttrList;
```

```

unreconciledList[1] = importedUnreconciledAttrList;
err = NvSciSyncAttrListReconcile(unreconciledList, 2, &reconciledList,
    &newConflictList);
if (err != NvSciError_Success) {
    goto fail;
}
/* ... */
NvSciSyncAttrListFree(importedUnreconciledAttrList);
NvSciSyncAttrListFree(reconciledList);

```

6.6.5.6 Sync Management

6.6.5.6.1 NvSciSync Objects

The NvSciSyncObj structure represents a sync object. It can be allocated after successful reconciliation. The reconciled attribute list contains all information about resources needed for the allocation.

6.6.5.6.1.1 NvSciSyncObj

```

/* Create NvSciSync object and get the syncObj */
NvSciError err;
err = NvSciSyncObjAlloc(reconciledList, &syncObj);
if (err != NvSciError_Success) {
    goto fail;
}
/* using the object, sharing it with others */
NvSciSyncAttrListFree(reconciledList);
NvSciSyncObjFree(syncObj);

```

6.6.5.6.2 Inter-Application

In the cross-process case, the reconciled list and object must be shared with all the waiters.

6.6.5.6.2.1 Export/Import NvSciSyncObj

```

/* signaller */
void* objAndList;
size_t objAndListSize;
NvSciError err;
err = NvSciSyncIpcExportAttrListAndObj(
    syncObj, /* syncObj to be exported
(the reconciled list is inside it) */
    NvSciSyncAccessPerm_WaitOnly, /* permissions we want the
receiver to have. Setting this to NvSciSyncAccessPerm_Auto allows
NvSciSync to automatically determine necessary permissions */
    ipcEndpoint, /* IpcEndpoint via which the
object is to be exported */
    &objAndList, /* descriptor of the object
and list to be communicated */
    &objAndListSize); /* size of the descriptor */
/* send via Ipc */
NvSciSyncAttrListAndObjFreeDesc(objAndList);

```

```

/* waiter */
void* objAndList;
size_t objAndListSize;
err = NvSciSyncIpcImportAttrListAndObj(
    module,                                /* NvSciSyncModule use to create
original unreconciled lists in the waiter */
    ipcEndpoint,                            /* ipcEndpoint from which the
descriptor was received */
    objAndList,                             /* the descriptor of the sync obj and
associated reconciled attribute list received from the signaler */
    objAndListSize,                         /* size of the descriptor */
    &waiterAttrList,                        /* the array of original unreconciled
lists prepared in the waiter */
    1,                                     /* size of the array */
    NvSciSyncAccessPerm_WaitOnly,           /* permissions expected by the waiter.
Setting this to NvSciSyncAccessPerm_Auto allows NvSciSync to automatically
determine necessary permissions */
    10000U,                                /* Recommended timeout in microseconds.
Some primitives might require time to transport all needed resources. */
    &syncObj);                            /* sync object generated from the
descriptor on the waiter's side */
/* use the sync object, perhaps export it to more peers... */
NvSciSyncObjFree(syncObj);

```

6.6.5.6.3 Cpu Wait Contexts

NvSciSync can be used to wait on a fence from the CPU but it might require some additional resources to perform this wait. Allocating those resources is controlled by the application and encapsulated in the NvSciSyncCpuWaitContext structure. In the initialization phase a waiter allocates this structure. It can then be used to wait on any number of sync fences but it cannot be used from multiple threads at the same time:

6.6.5.6.3.1 NvSciSyncCpuWaitContext

```

/* waiter */
NvSciSyncCpuWaitContext waitContext = NULL;
NvSciError err;
/* initialize module */
err = NvSciSyncCpuWaitContextAlloc(module, &waitContext);
if (err != NvSciError_Success) {
    goto fail;
}
/* more initialization, using the context for fence waiting */
NvSciSyncCpuWaitContextFree(waitContext);

```

6.6.5.6.4 NvSciSyncFence Operations

After successfully allocating an object and exporting it to all the waiters, the application can proceed to the runtime phase. Typically, it is a loop where the signaler prepares its job, associates its completion with a sync fence, and shares the fence with a waiter. The waiter constructs its job in such a way that it only starts after the fence expires. Both waiter and signaler then enqueue their jobs, and the use of fences establishes ordering: the waiter's job only starts when the signaler's job is complete.

6.6.5.6.4.1 NvSciSyncFence Cpu operations

```

/* signaler*/
NvSciSyncFence sharedFence = NvSciSyncFenceInitializer;
NvSciSyncFence localFence = NvSciSyncFenceInitializer; /* always
initialize with the Initializer */
NvSciError err;
err = NvSciSyncObjGenerateFence(syncObj, &localFence);
if (err != NvSciError_Success) {
    goto fail;
}
/* duplicate fence before sharing */
err = NvSciSyncFenceDup(&localFence, sharedFence);
if (err != NvSciError_Success) {
    goto fail;
}
/* create more duplicates if necessary */
/* communicate the fence to the waiter. */
/* local copy no longer necessary, so dispose of it */
NvSciSyncFenceClear(&localFence); /* this call cleans references to
sync object and is needed for proper freeing */
/* do something else, like some CPU job */
err = NvSciSyncObjSignal(syncObj);
if (err != NvSciError_Success) {
    goto fail;
}
/* waiter */
/* receive the sharedFence from the signaller */
err = NvSciSyncFenceWait(sharedFence,
    waitContext, NV_WAIT_INFINITE);
if (err != NvSciError_Success) {
    return err;
}
NvSciSyncFenceClear(sharedFence);

```

6.6.5.6.5 Inter-Application

The above examples assume an inter thread case but in an inter process case the fence must be exported and imported, similar to how the attribute lists were packaged.

6.6.5.6.5.1 Export/Import NvSciSyncFence

```

/* signaller*/
NvSciSyncFenceIpcExportDescriptor fenceDesc;
NvSciError err;
/* generate sharedFence */
err = NvSciSyncIpcExportFence(
    &sharedFence,          /* fence to be exported */
    ipcEndpoint,           /* should be the same ipcEndpoint used for
communicating the attribute lists */
    &fenceDesc);          /* fence descriptor has a fixed size and
is only filled in this call */
if (err != NvSciError_Success) {
    return err;
}

```

```

NvSciSyncFenceClear(&sharedFence);
/* send the descriptor via Ipc */
/* waiter */
/* receive the descriptor fenceDesc */
err = NvSciSyncIpcImportFence(syncObj,
                             fenceDesc,
                             &syncFence);
if (err != NvSciError_Success) {
    return err;
}

```

Fences are designed to be small, fixed sized objects, and interactions with them do not involve any runtime allocation. All fence structures and fence descriptors are allocated once at initialization. During runtime, NvSciSync only updates the fence and related structures, as needed.

6.6.5.7 Timestamps

NvSciSync supports timestamps in fences. They represent the time of a fence's expiration. This can help profiling the timing of streaming and debugging performance issues. This feature can be enabled during the initialization of the sync object. Then the waiter can call NvSciSyncFenceGetTimestamp to obtain the timestamp data in the streaming phase.

To enable this feature, the waiter should set

`NvSciSyncAttrKey_WaiterRequireTimestamps` to true in its attribute list.

6.6.5.7.1 Waiter requires timestamp

```

bool requireTimestamps = true;
NvSciSyncAttrKeyValuePair keyValue[] = {
    ... // other attributes
    { .attrKey = NvSciSyncAttrKey_WaiterRequireTimestamps,
      .value = (void*) &requireTimestamps,
      .len = sizeof(bool),
    },
};

```

During reconciliation, if the signaler supports timestamp, this feature is enabled. If the waiter doesn't require timestamp, then this feature is disabled. If the waiter requires a timestamp but the signaler doesn't support it, then the reconciliation fails.

During the streaming phase, the waiter can obtain the timestamp value by calling `NvSciSyncFenceGetTimestamp` on an expired fence.

6.6.5.7.2 Waiter gets timestamp

```

uint64_t timestamp;
err = NvSciSyncFenceWait(&fence, waitContext, NV_WAIT_INFINITE);
if (err != NvSciError_Success) {
    return err;
}

```

```

err = NvSciSyncFenceGetTimestamp(&fence, &timestamp);
if (err != NvSciError_Success) {
    return err;
}

```

6.6.5.8 Task Status in Fences

NvSciSync supports task status in fences. It provides the status of the task associated with the fence expiration. After waiting on a fence, a waiter can ask for task status with `NvSciSyncFenceGetTaskStatus()`.

```

NvSciSyncTaskStatus taskStatus;
err = NvSciSyncFenceWait(&fence, waitContext, NV_WAIT_INFINITE);
if (err != NvSciError_Success) {
    return err;
}
err = NvSciSyncFenceGetTaskStatus(&fence, &taskStatus);
if (err != NvSciError_Success) {
    return err;
}

```

6.6.5.9 UMD Access

6.6.5.9.1 NvMedia

NvMedia provides a set of interfaces to submit tasks to each of the hardware engines it supports. For example, the NvMedia2D* set of APIs provide the functionality to submit tasks to the VIC engine. Similarly, the NvMediaISP* set of APIs provide the functionality to submit tasks to the ISP hardware engine. An engine specific set of APIs are extended to support NvSciSync. They provide the following functionalities:

- > Takes in an NvSciSyncFence as input when the engine acts as a waiter.
- > Gives out an NvSciSyncFence as output when the engine acts as a signaler.

The following section uses NvMedia2D-NvSciSync APIs to demonstrate the usage of NvMedia-NvSciSync APIs. A similar set of APIs for other engines can be used in the same way.

6.6.5.9.1.1 Definitions

EOF Fence: End of frame fence. A fence whose expiry indicates that the output image is written.

PREFence: Start of engine operation is blocked until this fence expires.

6.6.5.9.2 NvMedia2D-NvSciSync

- > Query NvSciSyncObj attributes (for waiting or signaling) from NvMedia2D

Use `NvMedia2DFillNvSciSyncAttrList` API to query the NvSciSync attributes from NvMedia2D. NvSciSync objects allocated with such NvSciSyncAttrLists are only accepted by NvMedia2D-NvSciSync APIs.

- > NvSciSync object registration/unregistration with NvMedia2D.

Use NvMedia2DRegisterNvSciSyncObj API to register the NvSciSync objects with NvMedia2D. Every NvSciSyncObj used by NvMedia2D must be registered upfront with NvMedia2D.

During tear down, use NvMedia2DUnRegisterNvSciSyncObj API to unregister the registered NvSciSyncObjs with NvMedia2D.

- > Set NvSciSyncObj for end of frame (EOF) event usage with NvMedia2D.
- > Use NvMedia2DSetNvSciSyncObjforEOF API to tell NvMedia2D which NvSciSyncObj to use to signal the EOF event of the NvMedia2DBlitEx operation. A NvSciSyncObj must be set before calling NvMedia2DBlitEx API.
- > Wait for an NvSciSyncFence.
- > Get an NvSciSyncFence.

Use NvMedia2DGetEOFNvSciSyncFence API to get an NvSciSyncFence whose expiry indicates that the last submitted NvMedia2DBlitEx task has completed. NvMedia2DGetEOFNvSciSyncFence API can be called only after an NvMedia2DBlitEx API call.

Use NvMedia2DInsertPreNvSciSyncFence API to tell NvMedia2D to wait on a NvSciSyncFence before actually starting the VIC engine to work on the task submitted by NvMedia2DBlitEx API.

6.6.5.9.2.1 NvMedia2D NvSciSync API Usage

```
/* ***** Init-time *****/
NvSciSyncModule    nvscisyncModule;
NvSciError         nverr;
NvSciSyncAttrList  nvscisyncattr_w;
NvSciSyncAttrList  nvscisyncattr_s;
NvSciSyncAttrList  nvscisyncattr_unreconciled_h[2];
NvSciSyncAttrList  nvscisyncattr_reconciled;
NvSciSyncAttrList  ConflictAttrList;
NvSciSyncFence     eofnvscisyncfence = NV_SCI_SYNC_FENCE_INITIALIZER;
NvSciSyncObj       nvscisyncEOF, nvscisyncpre;
nvm2dhd़ = NvMedia2DCreate(nvmdevice);
nverr = NvSciSyncModuleOpen(&nvscisyncModule);
/*********NvMedia 2D as signaler *****/
nverr = NvSciSyncAttrListCreate(nvscisyncModule, &nvscisyncattr_s);
nvmstatus = NvMedia2DFillNvSciSyncAttrList(nvscisyncattr_s, NVMEDIA_SIGNALER);
nvscisyncattr_unreconciled_h[0] = nvscisyncattr_s;
nvscisyncattr_unreconciled_h[1] = get attribute list from the appropriate waiter;
nverr = NvSciSyncAttrListReconcile(nvscisyncattr_unreconciled_h[],
                                    2 , &nvscisyncattr_reconciled, &ConflictAttrList);
nverr = NvSciSyncObjAlloc(nvscisyncattr_reconciled, &nvscisyncEOF);
/*********NvMedia 2D as waiter *****/
nverr = NvSciSyncAttrListCreate(&nvscisyncattr_w);
nvmstatus = NvMedia2DFillNvSciSyncAttrList(nvscisyncattr_w, NVMEDIA_WAITER);
/*If the signaller is also in the same process as the 2D Waiter, then
NvSciSyncAttrListReconcileAndObjAlloc or NvSciSyncAttrListReconcile and
NvSciSyncObjAlloc API pair has/have to be used to allocate nvscisyncpre NvSciScynObject.
If the signaller is in a different process/VM than the 2D Waiter, then
```

```

NvSciSyncAttrList export/import APIs and NvSciSyncObjIpc Export/Import APIs
have to be used allocate a NvSciSyncObject on signaler and waiter sides.
nvscisyncpre is the imported NvSciSyncObject on the waiter side */
/*All the NvSciSyncObjects(NvSciSyncObjects associated with PreFences, EOFence
) which will be used by NvMedia2D must be registered upfront. */
/* *****Start of Registration of NvSciSync Objects *****/
nvmstatus = NvMedia2DRegisterNvSciSyncObj(nvm2dhd1, NVMEDIA_EOFSYNCOBJ, nvscisyncEOF);
/* Register all the NvSciSync objects which will be used to generate preferences for
NvMedia2DBlit operation. nvscisyncpre is one such Pre NvSciSync object */
nvmstatus = NvMedia2DRegisterNvSciSyncObj(nvm2dhd1, NVMEDIA_PRESYNCOBJ, nvscisyncpre);
*****End of Registration of NvSciSync Objects *****
/*Allocate a NvMediaImage for input, say inputimg */
/*Allocate a NvMediaImage for output, say outputimg */
*****End of Init-time and Start of Run-time *****
nvmstatus = NvMedia2DSetNvSciSyncObjforEOF(nvm2dhd1, nvscisyncEOF);
/*Get a nvscisyncfence from somewhere(maybe a eofnvscisyncfence of
some other engine operation) which needs to be inserted as preference
for 2DBlit operation. prenvscisyncfence is one such NvSciSyncFence. */
nvmstatus = NvMedia2DInsertPreNvSciSyncFence(nvm2dhd1, prenvscisyncfence);
nvmstatus = NvMedia2DBlitEx(nvm2dhd1, outputimg, NULL, inputimg, NULL,
                           2dblitparams, paramsout);
nvmstatus = NvMedia2DGetEOFNvSciSyncFence(nvm2dhd1, nvscisyncEOF, &eofnvscisyncfence);
/*eofnvscisyncfence may be used as preference for some other engine operation
or application can decide to wait on CPU till their expiry using NvSciSyncWait API. */
/* ***** End of Run time *****
/*Unregister all of the registered NvSciSync objects */
nvmstatus = NvMedia2DUnRegisterNvSciSyncObj(nvm2dhd1, nvscisyncEOF);
nvmstatus = NvMedia2DUnRegisterNvSciSyncObj(nvm2dhd1, nvscisyncpre);
NvSciSyncAttrListFree(nvscisyncattr_w);
NvSciSyncAttrListFree(nvscisyncattr_s);
NvSciSyncAttrListFree(nvscisyncattr_reconciled);
NvSciSyncObjFree(nvscisyncEOF);
NvSciSyncObjFree(nvscisyncpre);
NvSciSyncModuleClose(nvscisyncModule);

```

6.6.5.9.3 cuDLA

cuDLA supports NvSciSync by enabling applications to signal and wait for them. cuDLA treats NvSciSync as an external semaphore object, which can be imported into the DLA address space. The application can use the existing `cudlaImportExternalSemaphore` API to build dependencies between an NvSciSync object and cuDLA tasks, and vice-versa.

6.6.5.9.3.1 Importing a NvSciSync Object into cuDLA

1. Query NvSciSyncObj attributes (for waiting or signaling) from cuDLA.

Use `cudlaGetNvSciSyncAttributes` API to query the NvSciSync attributes from cuDLA for a given cuDLA device. NvSciSyncAttrLists passed to this API must be created and managed by the application.

2. NvSciSync objects registration/unregistration with cuDLA.

- > Use `cudlaImportExternalSemaphore` API to register/import an NvSciSync object into the DLA address space. This API accepts a valid NvSciSyncObject as a parameter to `cudlaExternalSemaphoreHandleDesc`.

- > On completion, the pointer returned by the above API internally holds a reference to the NvSciSyncObject passed earlier and can be used during DLA task submission to create wait events or signal events accordingly.
 - > Use `cudlaMemUnregister` to unregister/destroy an already registered/imported NvSciSync Object from the DLA address space.
3. Wait for an NvSciSyncFence.

All events that the application wishes to wait on must be specified in the `waitEvents` field for a particular task. The corresponding task would wait for all the wait dependencies to be met before beginning execution on the DLA HW. Such wait happens asynchronously on the DLA (i.e., the calling thread returns immediately).

4. Get an NvSciSyncFence.

- > `cudlaSignalEvents` takes a valid NvSciSyncFence. This is passed in `cudlaTask` when the task is triggered.
- > Upon return, the fence tracks the completion of all work submitted on that cuDLA task.
- > Waiting on a fence is equivalent to waiting for the completion of that cuDLA task. The NvSciSyncFence is signaled by the DLA when the task finishes, and any potential waiters waiting on the NvSciSyncFence are unblocked.



Note: Previous content of the passed NvSciSyncFence will be overwritten.

6.6.5.9.3.2 SampleUsage

```
/*********************Initialize NvSciSync
Parameters********************/
NvSciSyncObj syncObj1, syncObj2;
NvSciSyncModule syncModule;
NvSciSyncAttrList syncAttrListObj1[2];
NvSciSyncAttrList syncAttrListObj2[2];
NvSciSyncCpuWaitContext nvSciCtx;
NvSciSyncAttrList waiterAttrListObj1 = NULL;
NvSciSyncAttrList signalerAttrListObj1 = NULL;
NvSciSyncAttrList waiterAttrListObj2 = NULL;
NvSciSyncAttrList signalerAttrListObj2 = NULL;
NvSciSyncAttrList nvSciSyncConflictListObj1;
NvSciSyncAttrList nvSciSyncReconciledListObj1;
NvSciSyncAttrList nvSciSyncConflictListObj2;
NvSciSyncAttrList nvSciSyncReconciledListObj2;

NvSciSyncModuleOpen(&syncModule);

// Create Attribute list for NvSciSyncObj1
NvSciSyncAttrListCreate(syncModule, &signalerAttrListObj1);

NvSciSyncAttrListCreate(syncModule, &waiterAttrListObj1);

cudlaGetNvSciSyncAttributes(reinterpret_cast<uint64_t*>(waiterAttrListObj1),
CUDLA_NVSCISYNC_ATTR_WAIT);
```

```

// Fill CPU signaller Attribute list for NvSciSyncObj1 here
{
    bool cpuSignaler = true;
    NvSciSyncAttrKeyValuePair keyValue[2];
    memset(keyValue, 0, sizeof(keyValue));
    keyValue[0].attrKey = NvSciSyncAttrKey_NeedCpuAccess;
    keyValue[0].value = (void*) &cpuSignaler;
    keyValue[0].len = sizeof(cpuSignaler);

    NvSciSyncAccessPerm cpuPerm = NvSciSyncAccessPerm_SignalOnly;
    keyValue[1].attrKey = NvSciSyncAttrKey_RequiredPerm;
    keyValue[1].value = (void*) &cpuPerm;
    keyValue[1].len = sizeof(cpuPerm);

    NvSciSyncAttrListSetAttrs(signalerAttrListObj1, keyValue, 2);
}

// Reconcile attribute list for NvSciSyncObj1
syncAttrListObj1[0] = signalerAttrListObj1;
syncAttrListObj1[1] = waiterAttrListObj1;
NvSciSyncAttrListReconcile(syncAttrListObj1, 2, &nvSciSyncReconciledListObj1,
&nvSciSyncConflictListObj1);

// Allocate NvSciSyncObj1 here
NvSciSyncObjAlloc(nvSciSyncReconciledListObj1, &syncObj1);

NvSciSyncCpuWaitContextAlloc(syncModule, &nvSciCtx);

// Create Attribute list for NvSciSyncObj2
NvSciSyncAttrListCreate(syncModule, &signalerAttrListObj2);

NvSciSyncAttrListCreate(syncModule, &waiterAttrListObj2);

cudlaGetNvSciSyncAttributes(reinterpret_cast<uint64_t*>(signalerAttrListObj2), CUDLA_NVSCISYNC_ATT

// Fill CPU signaller Attribute list for NvSciSyncObj1 here
{
    bool cpuWaiter = true;
    NvSciSyncAttrKeyValuePair keyValue[2];
    memset(keyValue, 0, sizeof(keyValue));
    keyValue[0].attrKey = NvSciSyncAttrKey_NeedCpuAccess;
    keyValue[0].value = (void*) &cpuWaiter;
    keyValue[0].len = sizeof(cpuWaiter);

    NvSciSyncAccessPerm cpuPerm = NvSciSyncAccessPerm_WaitOnly;
    keyValue[1].attrKey = NvSciSyncAttrKey_RequiredPerm;
    keyValue[1].value = (void*) &cpuPerm;
    keyValue[1].len = sizeof(cpuPerm);

    NvSciSyncAttrListSetAttrs(waiterAttrListObj2, keyValue, 2);
}

// Reconcile attribute list for NvSciSyncObj1

```

```

syncAttrListObj2[0] = signallerAttrListObj2;
syncAttrListObj2[1] = waiterAttrListObj2;
NvSciSyncAttrListReconcile(syncAttrListObj2, 2, &nvSciSyncReconciledListObj2,
&nvSciSyncConflictListObj2);

// Allocate NvSciSyncObj1 here
NvSciSyncObjAlloc(nvSciSyncReconciledListObj2, &syncObj2);

*****Registration of NvSciSync with
cuDLA*****
uint64_t* nvSciSyncObjRegPtr1 = NULL;
uint64_t* nvSciSyncObjRegPtr2 = NULL;

cudaExternalSemaphoreHandleDesc semaMemDesc = { 0 };
// Fill up cudaExternalSemaphoreHandleDesc
memset(&semaMemDesc, 0, sizeof(semaMemDesc));
semaMemDesc.extSyncObject = syncObj1;
// Import NvSciSync objects into cuDLA
cudaImportExternalSemaphore(cudlaDevHandle, &semaMemDesc, &nvSciSyncObjRegPtr1, 0);

// Fill up cudaExternalSemaphoreHandleDesc
memset(&semaMemDesc, 0, sizeof(semaMemDesc));
semaMemDesc.extSyncObject = syncObj2;
// Import NvSciSync objects into cuDLA
cudaImportExternalSemaphore(cudlaDevHandle, &semaMemDesc, &nvSciSyncObjRegPtr2, 0);

// Create Wait events for which cuDLA is waiter
NvSciSyncFence preFence = NvSciSyncFenceInitializer;
NvSciSyncObjGenerateFence(syncObj1, &preFence);
cudlaWaitEvents* waitEvents;
waitEvents = (cudlaWaitEvents *)malloc(sizeof(cudlaWaitEvents));
waitEvents->numEvents = 1;
CudlaFence* preFences = (CudlaFence *)malloc(waitEvents->numEvents *
sizeof(CudlaFence));
preFences[0].fence = &preFence;
preFences[0].type = CUDLA_NVSCISYNC_FENCE;
waitEvents->preFences = preFences;

// Create Signal events for which cuDLA is signaller
cudlaSignalEvents* signalEvents;
signalEvents = (cudlaSignalEvents *)malloc(sizeof(cudlaSignalEvents));
signalEvents->numEvents = 1;
uint64_t** devPtrs = (uint64_t **)malloc(signalEvents->numEvents * sizeof(uint64_t
*));
devPtrs[0] = nvSciSyncObjRegPtr2;
signalEvents->devPtrs = devPtrs;
NvSciSyncFence eofFence = NvSciSyncFenceInitializer;
signalEvents->eofFences = (CudlaFence *)malloc(signalEvents->numEvents *
sizeof(CudlaFence));
signalEvents->eofFences[0].fence = &eofFence;
signalEvents->eofFences[0].type = CUDLA_NVSCISYNC_FENCE;

***** Task Submission to DLA
*****

```

```

// Creation of Task with wait and Signal events and submit it to DLA
cudlaTask task;
task.moduleHandle = moduleHandle;
task.outputTensor = &outputBufObjRegPtr; // DLA will write results into this
memory on completion of the task
task.numOutputTensors = 1;
task.numInputTensors = 1;
task.inputTensor = &inputBufObjRegPtr; // DLA will read the input data from this
memory.
task.waitEvents = waitEvents;
task.signalEvents = signalEvents;
cudlaSubmitTask(cudlaDevHandle, &task, 1, NULL, 0);

*****Signalling of NvSciSyncObj1 from
CPU*****
NvSciSyncObjSignal(syncObj1);

*****Waiting on NvSciSyncObj2 waiter is
cuDLA*****
NvSciSyncFenceWait(reinterpret_cast<NvSciSyncFence*>(signalEvents-
>eofFences[0].fence), nvSciCtx, -1);

*****Tear Down phase for
NvSciSync*****
// Unregister nvSciSync from cuDLA
cudlaMemUnregister(devHandle, nvSciSyncObjRegPtr1);
cudlaMemUnregister(devHandle, nvSciSyncObjRegPtr2);

// Free NvSciSync
vSciSyncObjFree(syncObj1);
NvSciSyncObjFree(syncObj2);
NvSciSyncAttrListFree(signalerAttrListObj1);
NvSciSyncAttrListFree(waiterAttrListObj1);
NvSciSyncAttrListFree(signalerAttrListObj2);
NvSciSyncAttrListFree(waiterAttrListObj2);
NvSciSyncAttrListFree(nvSciSyncConflictListObj1);
NvSciSyncAttrListFree(nvSciSyncReconciledListObj1);
NvSciSyncAttrListFree(nvSciSyncConflictListObj2);
NvSciSyncAttrListFree(nvSciSyncReconciledListObj2);
NvSciSyncCpuWaitContextFree(nvSciCtx);
NvSciSyncModuleClose(syncModule);
free(waitEvents);
free(preFences);
free(signalEvents->eofFences);
free(signalEvents);
free(devPtrs);
NvSciSyncFenceClear(&preFence);
NvSciSyncFenceClear(&eofFence);

```

6.6.5.9.4 CUDA

CUDA supports NvSciSync by enabling applications to signal and wait for them on a CUDA stream. (Signaling an NvSciSync is similar to `cudaEventRecord` and waiting for an NvSciSync is similar to issuing `cudaStreamWaitEvent`). CUDA treats NvSciSync as an external semaphore object of type `cudaExternalSemaphoreHandleType`, which can be imported into the CUDA address space. The application can use existing `cudaExternalSemaphore` API to build dependencies between an NvSciSync object and CUDA streams, and vice-versa. Since `cudaExternalSemaphore` APIs are treated as regular stream operations, CUDA-NvSciSync interop follows regular stream semantics.

6.6.5.9.4.1 CUDA APIs

Query NvSciSyncObj attributes (for waiting or signaling) from CUDA

Use `cudaDeviceGetNvSciSyncAttributes` API to query the NvSciSync attributes from CUDA for a given CUDA device. NvSciSyncAttrLists passed to this API must be allocated and managed by the application.

NvSciSync object registration/unregistration with CUDA

Use `cudalImportExternalSemaphore` API to register/import an NvSciSync Objects into the CUDA address space. This API accepts a valid NvSciSyncObject as a parameter to `semHandleDesc`. On completion, the `extSem_out` returned internally holds a reference to the NvSciSyncObject passed earlier and must be sent to the APIs listed below.

Use `cudaDestroyExternalSemaphore` (for runtime) to unregister/destroy an already registered/imported NvSciSync Object from the CUDA address space.

Wait for an NvSciSyncFence.

Use `cudaWaitExternalSemaphoresAsync` to make all operations enqueued on the CUDA stream (passed as a parameter to this API) wait until the NvSciSyncFence (sent as a parameter to this API via `paramsArray`) is signaled by the relevant signaler. Such a wait happens asynchronously on the GPU (i.e., the calling thread returns immediately). Applications can also optionally set flag `CUDA_EXTERNAL_SEMAPHORE_WAIT_SKIP_NVSCIBUF_MEMSYNC` to indicate that memory synchronization operations are disabled over all CUDA-NvSciBufs imported into CUDA (in that process), which are normally performed by default to ensure data coherency with other importers of the same NvSciBuf memory objects. Use this flag when CUDA-NvSciSync is used to build only control-dependencies (i.e., no data sharing between the signaller and waiter).

Get an NvSciSyncFence.

`cudaSignalExternalSemaphoresAsync` takes a valid NvSciSyncFence as input. Upon return, the fence tracks the completion of all work submitted to the same CUDA stream on which the API was invoked. Waiting on a fence is equivalent to waiting for the completion of all the work on the stream. This API ensures that when the dependent work (in the stream) completes, the NvSciSyncFence is signaled, and any potential waiters waiting on the NvSciSyncFence are unblocked. The signal happens asynchronously in the GPU (i.e., the calling thread returns immediately). Applications can also optionally set flag `CUDA_EXTERNAL_SEMAPHORE_SIGNAL_SKIP_NVSCIBUF_MEMSYNC` to indicate that memory synchronization operations are disabled over all CUDA-NvSciBufs

imported into CUDA (in that process), which are normally performed by default to ensure data coherency with other importers of the same NvSciBuf memory objects. Use this flag when CUDA-NvSciSync is used to build only control-dependencies (i.e., no data sharing between the signaller and waiter).


Note:

`cudaWait|SignalExternalSemaphoresAsync` API takes an array of `cudaExternalSemaphore_t` and `cudaExternalSemaphoresWait|SignalParams`. This allows the application to enqueue one or more external semaphore objects, each being one of the `cudaExternalSemaphoreHandleType` types. This option is an efficient way to describe a dependency between a CUDA stream and more than one NvSciSyncFence as a single operation.

`cudaSignalExternalSemaphoresAsync` overwrites the previous contents of `NvSciSyncFence` passed to it.

CUDA-NvSciSync API Usage

```

NvSciSyncFence *signalerFence = NULL;
NvSciSyncFence *waiterFence = NULL;
NvSciIpcEndpoint signalerIpcEndpoint = 0;
NvSciIpcEndpoint waiterIpcEndpoint = 0;
NvSciSyncAttrList unreconciledList[2] = {NULL};
NvSciSyncAttrList reconciledList = NULL;
NvSciSyncAttrList newConflictList = NULL;
NvSciSyncAttrList signalerAttrList = NULL;
NvSciSyncAttrList waiterAttrList = NULL;
NvSciSyncAttrList importedWaiterAttrList = NULL;
NvSciSyncObjIpcExportDescriptor objDesc;
NvSciSyncFenceIpcExportDescriptor fenceDesc;
NvSciSyncObj signal0bj;
NvSciSyncObj wait0bj;
NvSciSyncModule module = NULL;
void* objAndList;
size_t objAndListSize = 0;
void* waiterListDesc;
size_t waiterAttrListSize = 0;
CUcontext signalerCtx = 0;
CUcontext waiterCtx = 0;
int iGPU = 0;
int dGPU = 1;
cudaStream_t signalerCudaStream;
cudaStream_t waiterCudaStream;
cudaExternalSemaphore_t signalerSema, waiterSema;
cudaExternalSemaphoreHandleDesc semaDesc;
cudaExternalSemaphoreSignalParams sigParams;
cudaExternalSemaphoreWaitParams waitParams;
/*********************INIT PHASE********************/
err = NvSciSyncModuleOpen(&module);
err = NvSciIpcInit();
err = NvSciIpcOpenEndpoint("ipc_test", &signalerIpcEndpoint);
err = NvSciIpcOpenEndpoint("ipc_test", &waiterIpcEndpoint);
err = NvSciSyncAttrListCreate(module, &signalerAttrList);

```

```

err = NvSciSyncAttrListCreate(module, &waiterAttrList);
signalerFence = (NvSciSyncFence *)calloc(1, sizeof(*signalerFence));
waiterFence = (NvSciSyncFence *)calloc(1, sizeof(*waiterFence));
cudaFree(0);
cudaSetDevice(iGPU); // Signaler will be on Device-1/iGPU
cuCtxCreate(&signalerCtx, CU_CTX_MAP_HOST, iGPU);
cudaSetDevice(dGPU); // Waiter will be on Device-0/dGPU
cuCtxCreate(&waiterCtx, CU_CTX_MAP_HOST, dGPU);
cuCtxPushCurrent(signalerCtx);
cudaStreamCreate(&signalerCudaStream);
cuCtxPopCurrent(&signalerCtx);
cuCtxPushCurrent(waiterCtx);
cudaStreamCreate(&waiterCudaStream);
cuCtxPopCurrent(&waiterCtx);
cuCtxPushCurrent(waiterCtx);
cudaDeviceGetNvSciSyncAttributes(waiterAttrList, dGPU, cudaNvSciSyncAttrWait);
err = NvSciSyncAttrListIpcExportUnreconciled(&waiterAttrList, 1, waiterIpcEndpoint,
    &waiterListDesc, &waiterAttrListSize);
// Allocate cuda memory for the signaler, if needed
cuCtxPopCurrent(&waiterCtx);
cuCtxPushCurrent(signalerCtx);
cudaDeviceGetNvSciSyncAttributes(signalerAttrList, iGPU, cudaNvSciSyncAttrSignal);
// Allocate cuda memory for the waiter, if needed
err = NvSciSyncAttrListIpcImportUnreconciled(module, signalerIpcEndpoint,
    waiterListDesc, waiterAttrListSize, &importedWaiterAttrList);
cuCtxPopCurrent(&signalerCtx);
unreconciledList[0] = signalerAttrList;
unreconciledList[1] = importedWaiterAttrList;
err = NvSciSyncAttrListReconcile(unreconciledList, 2, &reconciledList,
    &newConflictList);
err = NvSciSyncObjAlloc(reconciledList, &signal0bj);
// Export Created NvSciSyncObj and attribute list to waiter
err = NvSciSyncIpcExportAttrListAndObj(signal0bj, NvSciSyncAccessPerm_WaitOnly,
    signalerIpcEndpoint, &objAndList, &objAndListSize);
// Import already created NvSciSyncObj into a new NvSciSyncObj
err = NvSciSyncIpcImportAttrListAndObj(module, waiterIpcEndpoint, objAndList,
    objAndListSize, &waiterAttrList, 1, NvSciSyncAccessPerm_WaitOnly, 1000000, &wait0bj);
cuCtxPushCurrent(signalerCtx);
semaDesc.type = cudaExternalSemaphoreHandleTypeNvSciSync;
semaDesc.handle.nvSciSyncObj = (void*)signal0bj;
cudaImportExternalSemaphore(&signalerSema, &semaDesc);
cuCtxPopCurrent(&signalerCtx);
cuCtxPushCurrent(waiterCtx);
semaDesc.type = cudaExternalSemaphoreHandleTypeNvSciSync;
semaDesc.handle.nvSciSyncObj = (void*)wait0bj;
cudaImportExternalSemaphore(&waiterSema, &semaDesc);
cuCtxPopCurrent(&waiterCtx);
/*********************************************
*****STREAMING PHASE*****
*****
cuCtxPushCurrent(signalerCtx);
sigParams.params.nvSciSync.fence = (void*)signalerFence;
sigParams.flags = 0; //Set flags = cudaExternalSemaphoreSignalSkipNvSciBufMemSync if
needed
// LAUNCH CUDA WORK ON signalerCudaStream

```

```

cudaSignalExternalSemaphoresAsync(&signalerSema, &sigParams, 1, signalerCudaStream);
err = NvSciSyncIpcExportFence(signalerFence, signalerIpcEndpoint, &fenceDesc);
NvSciSyncFenceClear(signalerFence);
cuCtxPopCurrent(&signalerCtx);
cuCtxPushCurrent(waiterCtx);
err = NvSciSyncIpcImportFence(waitObj, &fenceDesc, waiterFence);
waitForParams.params.nvSciSync.fence = (void*)waiterFence;
waitForParams.flags = 0; //Set flags = cudaExternalSemaphoreWaitSkipNvSciBufMemSync if
needed
cudaWaitExternalSemaphoresAsync(&waiterSema, &waitForParams, 1, waiterCudaStream);
// LAUNCH CUDA WORK ON waiterCudaStream
cudaStreamSynchronize(waiterCudaStream);
cuCtxPopCurrent(&waiterCtx);
/*********************************************
*****TEAR-DOWN PHASE*****
*/
NvSciSyncObjFree(signalObj);
NvSciSyncObjFree(waitObj);
NvSciSyncAttrListFree(reconciledList);
NvSciSyncAttrListFree(newConflictList);
NvSciSyncAttrListFree(signalerAttrList);
NvSciSyncAttrListFree(waiterAttrList);
NvSciSyncAttrListFree(importedWaiterAttrList);
NvSciSyncModuleClose(module);
NvSciIpcCloseEndpoint(signalerIpcEndpoint);
NvSciIpcCloseEndpoint(waiterIpcEndpoint);
cudaStreamSynchronize(signalerCudaStream);
cudaStreamSynchronize(waiterCudaStream);
cudaStreamDestroy(waiterCudaStream);
cudaStreamDestroy(signalerCudaStream);
cudaDestroyExternalSemaphore(signalerSema);
cudaDestroyExternalSemaphore(waiterSema);
cuCtxDestroy(signalerCtx);
cuCtxDestroy(waiterCtx);
free(signalerFence);
free(waiterFence);
/*********************************************

```

6.6.5.9.5 Vulcan SC

Vulkan SC supports NvSciSync by enabling applications to signal and wait for them on a Vulkan SC synchronization object (VkFence and VkSemaphore). Vulkan SC can import NvSciSyncObj as an external synchronization object into its address space when applications enable the VK_NV_external_sci_sync2 extension. The application can use VkSemaphore to synchronize between NvSciSync object and Vulkan SC GPU submissions. Because VkFence and VkSemaphore are the existing synchronization primitive in Vulkan SC, Vulkan SC-NvSciSync interop follows regular Vulkan SC usage.

6.6.5.9.5.1 Query NvSciSyncObj Attributes from Vulkan SC

Use the vkGetPhysicalDeviceSciSyncAttributesNV API to query the NvSciSync attributes from Vulkan SC for a given Vulkan SC physical device. The NvSciSyncAttrLists passed to this API must be allocated and managed by application. Depending on the Vulkan SC synchronization primitive (VkFence or VkSemaphore) used,

`VkSciSyncAttributesInfoNV::VkSciSyncPrimitiveTypeNV` needs to be specified for selecting the correct primitive. Currently, `VkFence` only supports interop with sync point backed `NvSciSync` object (`NvSciSyncAttrValPrimitiveType_Syncpoint`), and `VkSemaphore` only supports interop with sysmem semaphore backed `NvSciSync` object (`NvSciSyncAttrValPrimitiveType_SysmemSemaphore`).

6.6.5.9.5.2 NvSciSync Object Import into Vulkan SC

Import NvSciSyncObj to VkFence

Use the `vkImportFenceSciSyncObjNV` API to import an `NvSciSync` object into Vulkan SC address space. This API takes a valid sync point-backed `NvSciSyncObj` as a parameter to `VkImportFenceSciSyncInfoNV::handle`. The application must specify `VkImportFenceSciSyncInfoNV::handleType` to `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV`.

Import NvSciSyncObj to VkSemaphore

Use the `vkCreateSemaphoreSciSyncPoolNV` API to import an `NvSciSync` object into the Vulkan SC address space. This API accepts a valid sysmem semaphore-backed `NvSciSyncObj` as a parameter to `VkSemaphoreSciSyncPoolCreateInfoNV::handle`. Once completed successfully, this API returns a valid `VkSemaphoreSciSyncPoolNV` object as a pool for `VkSemaphore` objects, because an `NvSciSyncObj` may represent more than one sysmem semaphore. Application can later retrieve the `VkSemaphore` object from the `VkSemaphoreSciSyncPoolNV` object.

For both `VkFence` and `VkSemaphore` imports, Vulkan SC will only create a new reference on the imported `NvSciSync` object; the application must use the `NvSciSync` API to destroy the `NvSciSync` object.

6.6.5.9.5.3 Wait for an NvSciSyncFence

VkFence (CPU Wait Only)

Use `vkImportFenceSciSyncFenceNV` to import an `NvSciSyncFence` to the `VkFence`. This API takes a valid `NvSciSyncFence` as a parameter to `VkImportFenceSciSyncInfoNV::handle`. The applications must specify `VkImportFenceSciSyncInfoNV::handleType` to `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV`. On completion, use the `vkWaitForFences` API to perform the CPU wait until the imported `NvSciSyncFence` is signaled by the relevant signaler or timeout occurs.

VkSemaphore

Use the `vkCreateSemaphore` API to retrieve a `VkSemaphore` object from the `VkSemaphoreSciSyncPoolNV` object according to the input `NvSciSyncFence`. This API takes a valid `VkSemaphoreCreateInfo` struct as a parameter. Application must chain a `VkSemaphoreSciSyncCreateInfoNV` struct to the `VkSemaphoreCreateInfo::pNext` with `VkSemaphoreSciSyncCreateInfoNV::semaphorePool` of a valid `VkSemaphoreSciSyncPoolNV` object and `VkSemaphoreSciSyncCreateInfoNV::pFence` of a valid `NvSciSyncFence`. In addition, application must chain `VkSemaphoreTypeCreateInfo` struct with `VkSemaphoreTypeCreateInfo::semaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` to the `VkSemaphoreCreateInfo::pNext`.

> GPU Wait

Use the `vkQueueSubmit` API to submit jobs enqueued on the Vulkan SC queue. This API takes `VkSubmitInfo::pWaitSemaphores` as a parameter. In addition, application must chain a valid `VkTimelineSemaphoreSubmitInfo` struct to the `VkSubmitInfo::pNext`, where `VkTimelineSemaphoreSubmitInfo::pWaitSemaphoreValues` can be retrieved using the NvSciSync API `NvSciSyncFenceExtractFence`. The jobs submitted to the Vulkan SC queue wait until the `NvSciSyncFence` is signaled by the relevant signaler. The wait occurs asynchronously on the GPU.

> CPU Wait

Use the `vkWaitSemaphores` API to wait `NvSciSyncFence` on the CPU. The application must provide a valid `VkSemaphore` as a parameter to `VkSemaphoreWaitInfo::pSemaphores` and a semaphore value extracted from `NvSciSyncFenceExtractFence` to `VkSemaphoreWaitInfo::pValues`. The wait is blocked on the CPU until the `NvSciSyncFence` is signaled by the relevant signaler or timeout occurs.

> Deterministic Fence

Vulkan SC supports Deterministic Fence in `VkSemaphore`. When Deterministic Fence is enabled in NvSciSync via `NvSciSyncAttrKey_RequireDeterministicFences`, Vulkan SC waiter does not need to import `NvSciSyncFence` from `NvSciIPC`. Instead, applications must always construct a new `NvSciSyncFence` by using 0 as the fence id. Increment the counter value by one from the previous semaphore wait operation at each semaphore wait operation. The initial counter value is 0.

6.6.5.9.5.4 Signal for an `NvSciSyncFence`

`VkFence` (CPU only)

Use the `vkQueueSubmit` API to submit jobs enqueued on the Vulkan SC queue by providing a valid `VkFence` to `fence` as a parameter. The fence tracks the completion of all the jobs submitted to the Vulkan SC queue on which the API was invoked.

Use `vkGetFenceSciSyncFenceNV` to get the pending `NvSciSyncFence`. The application must provide a valid `VkFence` to `VkFenceGetSciSyncInfoNV::fence` from which the `NvSciSyncFence` is extracted.

Waiting on this `NvSciSyncFence` is equivalent to waiting for the completion of all prior submissions on the queue, in submission order. Once all the submissions have completed, the `NvSciSyncFence` is signaled and all the potential waiters on this `NvSciSyncFence` are unblocked. This signal occurs asynchronously on the GPU.

`VkSemaphore`

> GPU Signal

Use the `vkQueueSubmit` API to submit jobs enqueued on the Vulkan SC queue. This API takes `VkSubmitInfo::pSignalSemaphores` as a parameter. In addition, application must chain a valid `VkTimelineSemaphoreSubmitInfo` struct to the `VkSubmitInfo::pNext`, where

`VkTimelineSemaphoreSubmitInfo::pSignalSemaphoreValues` should be tracked and maintained by application (The initial value is always 0).

As applications must maintain the signal value by incrementing it for each submission, use the NvSciSync API `NvSciSyncFenceUpdateFence` to get the pending `NvSciSyncFence`, where the id is always 0 because `VkSemaphore` only supports one sysmem semaphore as a signaler. This `NvSciSyncFence` tracks prior batches to the queue, in signal operation order. Once it is signaled, all the potential waiters on this `NvSciSyncFence` will be unblocked. This signal occurs asynchronously on the GPU.

> CPU Signal

Use the `vkSignalSemaphore` API to signal on the CPU. This API takes a valid `VkSemaphoreSignalInfo::semaphore` as a parameter. In addition, applications must track and maintain the signal value and provide it as a parameter to `VkSemaphoreSignalInfo::value`.

Use the NvSciSync API `NvSciSyncFenceUpdateFence` to get the pending `NvSciSyncFence`. The signal occurs asynchronously on the GPU. Once it is signaled, all the potential waiters on this `NvSciSyncFence` will be unblocked.

> Deterministic Fence

Vulkan SC supports Deterministic Fence in `VkSemaphore`. When Deterministic Fence is enabled in NvSciSync, Vulkan SC signaler doesn't need to export `NvSciSyncFence` to `NvSciIPC`. When Vulkan SC applications pass a counter value to signal a `VkSemaphore`, applications pass the value, which increments by +1 (exactly one), from the counter value of the previous semaphore signal operation on that `VkSemaphore`, in signal operation order. The initial counter value is 0.

6.6.5.9.5.5 Export NvSciSync from Vulkan SC

In the Vulkan SC Specification, `VK_NV_external_sci_sync2` defines a feature to export an `NvSciSyncObj`. In Vulkan SC terminology, “**export**” indicates the Vulkan SC driver allocates and manages the `NvSciSyncObj` instead of application. To keep the interfaces symmetrical with other NVIDIA DRIVE® OS UMDs, this feature is always disabled. Applications can use the `vkGetPhysicalDeviceFeatures2` API by passing a `VkPhysicalDeviceExternalSciSync2FeaturesNV` struct to query the enablement of this feature.

6.6.5.9.5.6 Vulkan SC-NvSciSync API Usage Examples

Following is example code to demonstrate practical usage of `VK_NV_external_sci_sync2`. The example is edited for brevity, to focus on the interoperability of Vulkan SC and NvSciSync, is incomplete for the complete Vulkan SC creation, and ignores the `NvSciIPC` process. Your application must check each `VkResult` and `NvSciError` and API function returns, and handle any error values.

VkFence

```
VkPhysicalDevice physdev = VK_NULL_HANDLE;
VkDevice dev = VK_NULL_HANDLE;
// Create VkPhysicalDevice
// ...
```

```

VkDeviceCreateInfo deviceInfo {};
// Enable VK_NV_external_sci_sync2 extension
std::vector<const char*> extNamesDev;
extNamesDev.push_back(VK_NV_EXTERNAL_SCI_SYNC_2_EXTENSION_NAME);
deviceInfo.enabledExtensionCount = (uint32_t)extNamesDev.size();
deviceInfo.ppEnabledExtensionNames = extNamesDev.data();
// Initialize other parts of deviceInfo
// ...
vkCreateDevice(physdev[0], &deviceInfo, nullptr, &dev);

// Initial Phase
NvSciSyncModule scisyncModule {nullptr};
NvSciSyncAttrList attrListWaiter;
NvSciSyncModuleOpen(&scisyncModule);

// Waiter creates NvSciSyncAttrList
NvSciSyncAttrListCreate(scisyncModule, &attrListWaiter);
VkSciSyncAttributesInfoNV vkSciSyncAttributesInfo = {
    .sType = VK_STRUCTURE_TYPE_SCI_SYNC_ATTRIBUTES_INFO_NV,
    .pNext = nullptr,
    .clientType = VK_SCI_SYNC_CLIENT_TYPE_WAITER_NV,
    .primitiveType = VK_SCI_SYNC_PRIMITIVE_TYPE_FENCE_NV
};
vkGetPhysicalDeviceSciSyncAttributesNV(
    physdev, &vkSciSyncAttributesInfo, *attrListWaiter);

// Signaler creates NvSciSyncAttrList
NvSciSyncAttrList attrListSignaler;
VkSciSyncAttributesInfoNV vkSciSyncAttributesInfo = {
    .sType = VK_STRUCTURE_TYPE_SCI_SYNC_ATTRIBUTES_INFO_NV,
    .pNext = nullptr,
    .clientType = VK_SCI_SYNC_CLIENT_TYPE_SIGNALER_NV,
    .primitiveType = VK_SCI_SYNC_PRIMITIVE_TYPE_FENCE_NV
};
vkGetPhysicalDeviceSciSyncAttributesNV(
    physdev, &vkSciSyncAttributesInfo, *attrListSignaler);

// Using NvSciIPC to import/export NvSciSyncAttrList
// ...
// Signaler reconcile the NvSciSyncAttrList
NvSciSyncAttrList reconciledList;
NvSciSyncAttrList unreconciledList[2] = {attrListWaiter, attrListSignaler};
NvSciSyncAttrList conflictList;
NvSciSyncAttrListReconcile(unreconciledList, 2, &reconciledList, &conflictList);

// Signaler allocate NvSciSyncObj and import to VkFence
NvSciSyncObj signalerSciSyncObj = nullptr;
VkFence signalerVkFence = VK_NULL_HANDLE;
NvSciSyncObjAlloc(reconciledList, &signalerSciSyncObj);
VkFenceCreateInfo fenceCreateInfo = {};
fenceCreateInfo.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
vkCreateFence(dev, &fenceCreateInfo, nullptr, &signalerVkFence);

```

```

VkImportFenceSciSyncInfoNV importSciSyncInfo = {
    .sType = VK_STRUCTURE_TYPE_IMPORT_FENCE_SCI_SYNC_INFO_NV;
    .pNext = nullptr;
    .fence = signalerVkFence;
    .handleType = VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV
    .handle = signalerSciSyncObj;
};

vkImportFenceSciSyncObjNV(dev, &importSciSyncInfo);
// Export signalerSciSyncObj via NvSciIPC
// ...

NvSciSyncObj waiterSciSyncObj;
// Waiter Import waiterSciSyncObj via NvSciIPC
// ...

// Import to VkFence
VkFence waiterVkFence;
VkFenceCreateInfo fenceCreateInfo = {};
fenceCreateInfo.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
vkCreateFence(dev, &fenceCreateInfo, nullptr,
    &signalerVkFence);VkImportFenceSciSyncInfoNV importSciSyncInfo = {
    .sType = VK_STRUCTURE_TYPE_IMPORT_FENCE_SCI_SYNC_INFO_NV;
    .pNext = nullptr;
    .fence = waiterVkFence;
    .handleType = VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV
    .handle = waiterSciSyncObj;
};
vkImportFenceSciSyncObjNV(dev, &importSciSyncInfo);

// Run-Time Phase
// Signaler signals VkFence
VkSubmitInfo submitInfo = {};
vkQueueSubmit(vksc_queue, 1, &submitInfo, signalerVkFence);
VkFenceGetSciSyncInfoNV getSciSyncInfo = {
    .sType = VK_STRUCTURE_TYPE_FENCE_GET_SCI_SYNC_INFO_NV;
    .pNext = nullptr;
    .fence = signalerVkFence,
    .handleType = VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV,
};
NvSciSyncFence signalerSciSyncFence {NvSciSyncFenceInitializer};
vkGetFenceSciSyncFenceNV(dev, &getSciSyncInfo, &signalerSciSyncFence);
// Export signalerSciSyncFence via NvSciIPC
// ...

// Waiter waits VkFence
NvSciSyncFence waiterSciSyncFence {NvSciSyncFenceInitializer};
// Waiter imports waiterSciSyncFence via NvSciIPC
// ...

VkImportFenceSciSyncInfoNV importSciSyncInfo = {
    .sType = VK_STRUCTURE_TYPE_IMPORT_FENCE_SCI_SYNC_INFO_NV;
    .pNext = nullptr;
    .fence = waiterVkFence;
    .handleType = VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV
}

```

```

    .handle = waiterSciSyncFence;
};

vkImportFenceSciSyncFenceNV(dev, &importSciSyncInfo);
uint64_t timeout = 1000000LLU;
vkWaitForFences(dev, 1, waiterVkFence, timeout);

// Deinit Phase
NvSciSyncAttrListFree(reconciledList);
NvSciSyncAttrListFree(conflictList);
NvSciSyncAttrListFree(attrListWaiter);
NvSciSyncAttrListFree(attrListSignaler);
NvSciSyncModuleClose(scisyncModule);
vkDestroyFence(waiterVkFence);
vkDestroyFence(signalerVkFence);
vkDestroyDevice(dev, nullptr);

```

VkSemaphore

```

VkPhysicalDevice physdev = VK_NULL_HANDLE;
VkDevice dev = VK_NULL_HANDLE;
// Create VkPhysicalDevice
// ...

VkDeviceCreateInfo deviceInfo {};
// Enable VK_NV_external_sci_sync2 extension
std::vector<const char*> extNamesDev;
extNamesDev.push_back(VK_NV_EXTERNAL_SCI_SYNC_2_EXTENSION_NAME);
deviceInfo.enabledExtensionCount = (uint32_t)extNamesDev.size();
deviceInfo.ppEnabledExtensionNames = extNamesDev.data();
// Initialize other parts of deviceInfo
// ...
vkCreateDevice(physdev[0], &deviceInfo, nullptr, &dev);

// Initial Phase
NvSciSyncModule scisyncModule {nullptr};
NvSciSyncAttrList attrListWaiter;
NvSciSyncModuleOpen(&scisyncModule);

// Waiter creates NvSciSyncAttrList
NvSciSyncAttrListCreate(scisyncModule, &attrListWaiter);
VkSciSyncAttributesInfoNV vkSciSyncAttributesInfo = {
    .sType = VK_STRUCTURE_TYPE_SCI_SYNC_ATTRIBUTES_INFO_NV,
    .pNext = nullptr,
    .clientType = VK_SCI_SYNC_CLIENT_TYPE_WAITER_NV,
    .primitiveType = VK_SCI_SYNC_PRIMITIVE_TYPE_SEMAPHORE_NV
};
vkGetPhysicalDeviceSciSyncAttributesNV(
    physdev, &vkSciSyncAttributesInfo, *attrListWaiter);

// Signaler creates NvSciSyncAttrList
NvSciSyncAttrList attrListSignaler;
VkSciSyncAttributesInfoNV vkSciSyncAttributesInfo = {
    .sType = VK_STRUCTURE_TYPE_SCI_SYNC_ATTRIBUTES_INFO_NV,
    .pNext = nullptr,

```

```

    .clientType = VK_SCI_SYNC_CLIENT_TYPE_SIGNALER_NV,
    .primitiveType = VK_SCI_SYNC_PRIMITIVE_TYPE_SEMAPHORE_NV
};

vkGetPhysicalDeviceSciSyncAttributesNV(
    physdev, &vkSciSyncAttributesInfo, *attrListWaiter);

// Using NvSciIPC to import/export NvSciSyncAttrList
// ...
// Signaler reconcile the NvSciSyncAttrList
NvSciSyncAttrList reconciledList;
NvSciSyncAttrList unreconciledList[2] = {attrListWaiter, attrListSignaler};
NvSciSyncAttrList conflictList;
NvSciSyncAttrListReconcile(unreconciledList, 2, &reconciledList,&conflictList);

// Signaler allocate NvSciSyncObj and import to VkSemaphoreSciSyncPoolNV
NvSciSyncObj signalerSciSyncObj;
NvSciSyncObjAlloc(reconciledList, &signalerSciSyncObj);
VkSemaphoreSciSyncPoolCreateInfoNV signalerInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_POOL_CREATE_INFO_NV;
    .pNext = nullptr;
    .handle = signalerSciSyncObj
};
VkSemaphoreSciSyncPoolNV signalerSemPool = VK_NULL_HANDLE;
vkCreateSemaphoreSciSyncPoolNV(dev, &signalerInfo, nullptr, &signalerSemPool);

// Export signalerSciSyncObj via NvSciIPC
// ...

NvSciSyncObj waiterSciSyncObj;
// Waiter Import waiterSciSyncObj via NvSciIPC
// ...
VkSemaphoreSciSyncPoolCreateInfoNV waiterInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_POOL_CREATE_INFO_NV;
    .pNext = nullptr;
    .handle = waiterSciSyncObj
};
VkSemaphoreSciSyncPoolNV waiterSemPool = VK_NULL_HANDLE;
vkCreateSemaphoreSciSyncPoolNV(dev, &waiterInfo, nullptr, &signalerSemPool);

// Run-time Phase
// Signaler signals VkSemaphore
uint32_t const fenceId = 0;
uint32_t signalValue = 0;

// Application tracks and maintains the signal value
signalValue = signalVal + 1U;
NvSciSyncFence signalerFence{};
NvSciSyncFenceUpdateFence(signalerSciSyncObj, fenceId, signalValue, &signalerFence);

// Retrieve VkSemaphore from the VkSemaphoreSciSyncPoolNV providing
// NvSciSyncFence

```

```

VkSemaphoreSciSyncCreateInfoNV semaphoreSciSyncInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_CREATE_INFO_NV,
    .pNext = nullptr,
    .semaphorePool = signalerSemPool,
    .pFence = signalerFence
};

// Only timeline semaphore supports NvSciSync interops
VkSemaphoreTypeCreateInfo semaphoreTypeInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO,
    .pNext = &semaphoreSciSyncInfo,
    .semaphoreType = VK_SEMAPHORE_TYPE_TIMELINE,
    .initialValue = 0      // initialValue must be 0 with NvSciSyncObj
};

VkSemaphoreCreateInfo semaphoreCreateInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO,
    .pNext = &semaphoreTypeInfo,
    .flags = 0, // reserved bit, must be zero
};
VkSemaphore signalerSemaphore;
vkCreateSemaphore(dev, &semaphoreCreateInfo, nullptr, &signalerSemaphore);

// GPU Signal
if (GPU_SIGNAL) {

    VkTimelineSemaphoreSubmitInfo semaphoreInfo = {
        .sType = VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO,
        .pNext = nullptr,
        .waitSemaphoreValueCount = 0,
        .pWaitSemaphoreValues = nullptr,
        .signalSemaphoreValueCount = 1,
        .pSignalSemaphoreValues = &signalValue
    };
    VkSubmitInfo submitInfo = {
        .sType = VK_STRUCTURE_TYPE_SUBMIT_INFO,
        .pNext = &semaphoreInfo,
        .waitSemaphoreCount = 0,
        .pWaitSemaphores = nullptr,
        .pWaitDstStageMask = nullptr,
        .commandBufferCount = 0,
        .pCommandBuffers = nullptr,
        .signalSemaphoreCount = 1,
        .pSignalSemaphores = &signalerSemaphore,
    };
    vkQueueSubmit(m_queue, 1, &submitInfo, nullptr);
}

else if (CPU_SIGNAL) {
    VkSemaphoreSignalInfo signalInfo = {
        .sType = VK_STRUCTURE_TYPE_SEMAPHORE_SIGNAL_INFO,
        .pNext = nullptr,
        .semaphore = signalerSemaphore,
        .value = signalValue
}

```

```

    };
    vkSignalSemaphore(dev, &signalInfo);
}
// Export signalerFence via NvSciIPC
// ...
// Recycle the VkSemaphore back to the pool
vkDestroySemaphore(dev, signalerSemaphore, nullptr);

// Waiter waits VkSemaphore
NvSciSyncFence waiterFence{};
// Import waiterFence via NvSciIPC
// ...

uint32_t const fenceId = 0;
uint32_t waitValue = 0;
NvSciSyncFenceExtractFence(&waiterFence, &fenceId, &waitValue);
// Retrive VkSemaphore from the VkSemaphoreSciSyncPoolNV providing
// NvSciSyncFence
VkSemaphoreSciSyncCreateInfoNV semaphoreSciSyncInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_CREATE_INFO_NV,
    .pNext = nullptr,
    .semaphorePool = waiterSemPool,
    .pFence = waiterFence
};

// Only timeline semaphore supports NvSciSync interops
VkSemaphoreTypeCreateInfo sempahoreTypeInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO,
    .pNext = &semaphoreSciSyncInfo,
    .semaphoreType = VK_SEMAPHORE_TYPE_TIMELINE,
    .initialValue = 0      // initialValue must be 0 with NvSciSyncObj
};
VkSemaphoreCreateInfo semaphoreCreateInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO,
    .pNext = &sempahoreTypeInfo,
    .flags = 0, // reserved bit, must be zero
};
VkSemaphore waiterSemaphore;
vkCreateSemaphore(dev, &semaphoreCreateInfo, nullptr, &waiterSemaphore);
if (GPU_WAIT) {

    VkTimelineSemaphoreSubmitInfo semaphoreInfo = {
        .sType = VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO,
        .pNext = nullptr,
        .waitSemaphoreValueCount = 1,
        .pWaitSemaphoreValues = &waitValue,
        .signalSemaphoreValueCount = 0,
        .pSignalSemaphoreValues = nullptr
    };
    VkSubmitInfo submitInfo = {
        .sType = VK_STRUCTURE_TYPE_SUBMIT_INFO,
        .pNext = &semaphoreInfo,
        .waitSemaphoreCount = 1,
        .pWaitSemaphores = &waiterSemaphore,
    };
}

```

```

        .pWaitDstStageMask = nullptr,
        .commandBufferCount = 0,
        .pCommandBuffers = nullptr,
        .signalSemaphoreCount = 0,
        .pSignalSemaphores = nullptr,
    };
    vkQueueSubmit(m_queue, 1, &submitInfo, nullptr);
}
else if (CPU_WAIT) {
    VkSemaphoreWaitInfo waitInfo = {
        .sType = VK_STRUCTURE_TYPE_SEMAPHORE_WAIT_INFO,
        .pNext = nullptr,
        .flags = 0,
        .semaphoreCount = 1,
        .pSemaphores = &waiterSemaphore,
        .pValues = &value
    };
    uint64_t timeout = 1000000LLU;
    vkWaitSemaphores(m_dev, &waitInfo, timeout);
}
// Recycle the VkSemaphore back to the pool
vkDestroySemaphore(dev, vkWaitSemaphores, nullptr);

```

VkSemaphore on Deterministic Fence

```

// Skip the Init-time part, assume Deterministic Fence is enabled
// ...

// Run-time Phase
// Signaler signals VkSemaphore
uint32_t const fenceId = 0;
uint32_t signalValue = 0;
while (true) {

    // Application tracks and maintains the signal value, it will always increment
    // by 1 on the signal value.
    signalValue = signalVal + 1U;
    NvSciSyncFence signalerFence{};
    NvSciSyncFenceUpdateFence(signalerSciSyncObj, fenceId, signalValue, &signalerFence);

    // Retreive VkSemaphore from the VkSemaphoreSciSyncPoolNV providing
    // NvSciSyncFence
    VkSemaphoreSciSyncCreateInfoNV semaphoreSciSyncInfo = {
        .sType = VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_CREATE_INFO_NV,
        .pNext = nullptr,
        .semaphorePool = signallerSemPool,
        .pFence = signalerFence
    };

    // Only timeline semaphore supports NvSciSync interops
    VkSemaphoreTypeCreateInfo sempahoreTypeInfo = {
        .sType = VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO,
        .pNext = &semaphoreSciSyncInfo,
        .semaphoreType = VK_SEMAPHORE_TYPE_TIMELINE,
    };

```

```

    .initialValue = 0      // initialValue must be 0 with NvSciSyncObj
};

VkSemaphoreCreateInfo semaphoreCreateInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO,
    .pNext = &semaphoreCreateInfo,
    .flags = 0, // reserved bit, must be zero
};
VkSemaphore signalerSemaphore;
vkCreateSemaphore(dev, &semaphoreCreateInfo, nullptr, &signalerSemaphore);

// Only show the GPU Signal path
VkTimelineSemaphoreSubmitInfo semaphoreInfo = {
    .sType = VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO,
    .pNext = nullptr,
    .waitSemaphoreValueCount = 0,
    .pWaitSemaphoreValues = nullptr,
    .signalSemaphoreValueCount = 1,
    .pSignalSemaphoreValues = &signalValue
};
VkSubmitInfo submitInfo = {
    .sType = VK_STRUCTURE_TYPE_SUBMIT_INFO,
    .pNext = &semaphoreInfo,
    .waitSemaphoreCount = 0,
    .pWaitSemaphores = nullptr,
    .pWaitDstStageMask = nullptr,
    .commandBufferCount = 0,
    .pCommandBuffers = nullptr,
    .signalSemaphoreCount = 1,
    .pSignalSemaphores = &signalerSemaphore,
};
vkQueueSubmit(m_queue, 1, &submitInfo, nullptr);
// Recycle the VkSemaphore back to the pool
vkDestroySemaphore(dev, signalerSemaphore, nullptr);
}

// Waiter waits VkSemaphore
uint32_t const fenceId = 0;
uint32_t waitValue = 0;
while(true) {
    // Applications constructs a NvSciSyncFence themselves instead of importing from
    // NvSciIPC

    waitValue = waitValue + 1U;
    NvSciSyncFence waiterFence{};
    NvSciSyncFenceUpdateFence(waiterSciSyncObj, fenceId, waitValue, &waiterFence);
    // Retreive VkSemaphore from the VkSemaphoreSciSyncPoolNV providing
    // NvSciSyncFence
    VkSemaphoreSciSyncCreateInfoNV semaphoreSciSyncInfo = {
        .sType = VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_CREATE_INFO_NV,
        .pNext = nullptr,
        .semaphorePool = waiterSemPool,
        .pFence = waiterFence
}

```

```

};

// Only timeline semaphore supports NvSciSync interops
VkSemaphoreTypeCreateInfo sempahoreTypeInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO,
    .pNext = &semaphoreSciSyncInfo,
    .semaphoreType = VK_SEMAPHORE_TYPE_TIMELINE,
    .initialValue = 0      // initialValue must be 0 with NvSciSyncObj
};
VkSemaphoreCreateInfo semaphoreCreateInfo = {
    .sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO,
    .pNext = &sempahoreTypeInfo,
    .flags = 0, // reserved bit, must be zero
};
VkSemaphore waiterSemaphore;
vkCreateSemaphore(dev, &semaphoreCreateInfo, nullptr, &waiterSemaphore);

VkTimelineSemaphoreSubmitInfo semaphoreInfo = {
    .sType = VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO,
    .pNext = nullptr,
    .waitSemaphoreValueCount = 1,
    .pWaitSemaphoreValues = &waitForValue,
    .signalSemaphoreValueCount = 0,
    .pSignalSemaphoreValues = nullptr
};
VkSubmitInfo submitInfo = {
    .sType = VK_STRUCTURE_TYPE_SUBMIT_INFO,
    .pNext = &semaphoreInfo,
    .waitSemaphoreCount = 1,
    .pWaitSemaphores = &waiterSemaphore,
    .pWaitDstStageMask = nullptr,
    .commandBufferCount = 0,
    .pCommandBuffers = nullptr,
    .signalSemaphoreCount = 0,
    .pSignalSemaphores = nullptr,
};
vkQueueSubmit(m_queue, 1, &submitInfo, nullptr);
// Recycle the VkSemaphore back to the pool
vkDestroySemaphore(dev, vkWaitSemaphores, nullptr);
}

```

6.6.5.10 Late Attach

NvSciSync provides a mechanism for an application to allocate an NvSciSyncObj that will be shared with other peers, without requiring each peer to have sent an unreconciled NvSciSyncAttrList during initial reconciliation. The constraints of late attach waiter peers, however, should be passed using unreconciled NvSciSyncAttrList during the reconciliation process by the application to ensure the NvSciSyncObj is allocated to satisfy the constraints of the late peers.

NvSciSync also allows remote peer NvSciIpcEndpoint to gain access to already allocated NvSciSyncObj, provided the already allocated NvSciSyncObj satisfies

the constraints of the unreconciled NvSciSyncAttrList, received from that peer, through the NvSciSyncObjAttachPeer() API. NvSciSync provides attribute keys such as NvSciSyncAttrKey_PeerLocationInfo, NvSciSyncAttrKey_PrimitiveInfo and NvSciSyncAttrKey_PeerHwEngineArray that needs to be specified in the proxy unreconciled NvSciSyncAttrList during initial reconciliation.


Note:

- > Detailed descriptions of the attribute keys and the APIs are in the *NVIDIA DRIVE® OS API Reference*.
- > Application needs to refer to User Mode Driver (UMD) documentation for details of how to specify the NvSciSyncAttrKey_PrimitiveInfo and NvSciSyncAttrKey_PeerHwEngineArray keys.

6.6.5.11 Sample Application

6.6.5.11.1 CPU Signaler Usage

```

NvSciSyncAttrList unreconciledList[2] = {NULL};
NvSciSyncAttrList reconciledList = NULL;
NvSciSyncAttrList newConflictList = NULL;
NvSciSyncAttrList signallerAttrList = NULL;
NvSciSyncModule module = NULL;
NvSciSyncObj syncObj = NULL;
NvSciSyncAttrList importedUnreconciledAttrList = NULL;
NvSciSyncFence syncFence = NvSciSyncFenceInitializer;
NvSciIpcEndpoint ipcEndpoint = 0;
NvSciSyncFenceIpcExportDescriptor fenceDesc;
void* waiterAttrListDesc;
size_t waiterAttrListSize;
void* objAndListDesc;
size_t objAndListSize;
NvSciSyncAttrKeyValuePair keyValue[2] = {0};
bool cpuSignaler = true;
NvSciSyncAccessPerm cpuPerm;
/* Initialize NvSciIpc */
err = NvSciIpcInit();
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciIpcOpenEndpoint("example", &ipcEndpoint);
if (err != NvSciError_Success) {
    goto fail;
}
/* Signaler Setup/Init phase */
/* Initialize the NvSciSync module */
err = NvSciSyncModuleOpen(&module);
if (err != NvSciError_Success) {
    goto fail;
}
/* create local attribute list */
err = NvSciSyncAttrListCreate(module, &signallerAttrList);

```

```

if (err != NvSciError_Success) {
    goto fail;
}
err = largs->fillSignalerAttrList(signalerAttrList);
if (err != NvSciError_Success) {
    goto fail;
}
cpuSignaler = true;
keyValue[0].attrKey = NvSciSyncAttrKey_NeedCpuAccess;
keyValue[0].value = (void*) &cpuSignaler;
keyValue[0].len = sizeof(cpuSignaler);
cpuPerm = NvSciSyncAccessPerm_SignalOnly;
keyValue[1].attrKey = NvSciSyncAttrKey_RequiredPerm;
keyValue[1].value = (void*) &cpuPerm;
keyValue[1].len = sizeof(cpuPerm);
err = NvSciSyncAttrListSetAttrs(list, keyValue, 2);
if (err != NvSciError_Success) {
    goto fail;
}
/* receive waiterAttrListSize; */
/* receive waiterAttrListDesc */
err = NvSciSyncAttrListIpcImportUnreconciled(
    module, ipcEndpoint,
    waiterAttrListDesc, waiterAttrListSize,
    &importedUnreconciledAttrList);
if (err != NvSciError_Success) {
    goto fail;
}
unreconciledList[0] = signalerAttrList;
unreconciledList[1] = importedUnreconciledAttrList;
/* Reconcile Signaler and Waiter NvSciSyncAttrList */
err = NvSciSyncAttrListReconcile(
    unreconciledList, 2, &reconciledList,
    &newConflictList);
if (err != NvSciError_Success) {
    goto fail;
}
/* Create NvSciSync object and get the syncObj */
err = NvSciSyncObjAlloc(reconciledList, &syncObj);
if (err != NvSciError_Success) {
    goto fail;
}
/* Export attr list and obj and signal waiter*/
err = NvSciSyncIpcExportAttrListAndObj(
    syncObj,
    NvSciSyncAccessPerm_WaitOnly, ipcEndpoint,
    &objAndListDesc, &objAndListSize);
/* send objAndListSize */
/* send objAndListDesc */
/* signaller's streaming phase */
err = NvSciSyncObjGenerateFence(syncObj, &syncFence);
if (err != NvSciError_Success) {
    return err;
}

```

```

err = NvSciSyncIpcExportFence(&syncFence, ipcEndpoint, &fenceDesc);
if (err != NvSciError_Success) {
    goto fail;
}
NvSciSyncFenceClear(&syncFence);
/* do job that the waiter is supposed to wait on */
NvSciSyncObjSignal(syncObj);
/* cleanup */
fail:
/* Free descriptors */
free(objAndListDesc);
free(waiterAttrListDesc);
/* Free NvSciSyncObj */
NvSciSyncObjFree(syncObj);
/* Free Attribute list objects */
NvSciSyncAttrListFree(reconciledList);
NvSciSyncAttrListFree(newConflictList);
NvSciSyncAttrListFree(signalerAttrList);
NvSciSyncAttrListFree(importedUnreconciledAttrList);
/* Deinitialize the NvSciSync module */
NvSciSyncModuleClose(module);
/* Deinitialize NvSciIpc */
NvSciIpcCloseEndpoint(ipcEndpoint);
NvSciIpcDeinit();

```

6.6.5.11.2 CPU Waiter Usage

```

NvSciSyncAttrKeyValuePair keyValue[2] = {0};
NvSciSyncModule module = NULL;
NvSciSyncAttrList waiterAttrList = NULL;
void* waiterAttrListDesc;
size_t waiterAttrListSize;
NvSciSyncObj syncObj = NULL;
void* objAndListDesc = NULL;
size_t objAndListSize = 0U;
NvSciSyncCpuWaitContext waitContext = NULL;
NvSciSyncFenceIpcExportDescriptor fenceDesc;
NvSciIpcEndpoint ipcEndpoint = 0;
bool cpuWaiter = true;
NvSciSyncAttrKeyValuePair keyValue[2] = {0};
NvSciSyncAccessPerm cpuPerm = NvSciSyncAccessPerm_WaitOnly;
err = NvSciIpcInit();
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciIpcOpenEndpoint("example", &ipcEndpoint);
if (err != NvSciError_Success) {
    goto fail;
}
/* Waiter Setup/Init phase */
/* Initialize the NvSciSync module */
err = NvSciSyncModuleOpen(&module);
if (err != NvSciError_Success) {

```

```

        goto fail;
    }
err = NvSciSyncCpuWaitContextAlloc(module, &waitContext);
if (err != NvSciError_Success) {
    goto fail;
}
/* Get waiter's NvSciSyncAttrList from NvSciSync for CPU waiter */
err = NvSciSyncAttrListCreate(module, &waiterAttrList);
if (err != NvSciError_Success) {
    goto fail;
}
cpuWaiter = true;
keyValue[0].attrKey = NvSciSyncAttrKey_NeedCpuAccess;
keyValue[0].value = (void*) &cpuWaiter;
keyValue[0].len = sizeof(cpuWaiter);
cpuPerm = NvSciSyncAccessPerm_WaitOnly;
keyValue[1].attrKey = NvSciSyncAttrKey_RequiredPerm;
keyValue[1].value = (void*) &cpuPerm;
keyValue[1].len = sizeof(cpuPerm);
err = NvSciSyncAttrListSetAttrs(list, keyValue, 2);
if (err != NvSciError_Success) {
    goto fail;
}
/* Export waiter's NvSciSyncAttrList */
err = NvSciSyncAttrListIpcExportUnreconciled(
    &waiterAttrList, 1,
    ipcEndpoint,
    &waiterAttrListDesc, &waiterAttrListSize);
if (err != NvSciError_Success) {
    goto fail;
}
/* send waiterAttrListSize */
/* send waiterAttrListDesc */
/* receive objAndListDesc */
err = NvSciSyncIpcImportAttrListAndObj(
    module, ipcEndpoint,
    objAndListDesc, objAndListSize,
    &waiterAttrList, 1,
    NvSciSyncAccessPerm_WaitOnly, 10000U, &syncObj);
if (err != NvSciError_Success) {
    goto fail;
}
/* Waiter streaming phase */
/* receive fenceDesc */
err = NvSciSyncIpcImportFence(
    syncObj,
    &fenceDesc,
    &syncFence);
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciSyncFenceWait(
    &syncFence,
    waitContext, 30000U);

```

```

if (err != NvSciError_Success) {
    goto fail;
}
NvSciSyncFenceClear(&syncFence);
/* cleanup */
fail:
free(waiterAttrListDesc);
free(objAndListDesc);
NvSciSyncAttrListFree(waiterAttrList);
NvSciSyncObjFree(syncObj);
NvSciSyncCpuWaitContextFree(waitContext);
/* Deinitialize the NvSciSync module */
NvSciSyncModuleClose(module);
/* Deinitialize NvSciIpc */
NvSciIpcCloseEndpoint(ipcEndpoint);
NvSciIpcDeinit();

```

6.6.5.12 NvSciSync Profiling

NvSciSync supports visualizing of NvSciSyncFence execution on Nsight systems.

The following information can be collected and presented on the timeline in the report:

- CPU Signaling of NvSciSyncObj
- CPU Waiting of NvSciSyncFence

The previous events are only supported for NvSciSyncObj(s), which has NvSciSyncAttrValPrimitiveType_Syncpoint as NvSciSyncAttrKey_PrimitiveInfo.

This feature is supported on AV+L standard, AV+Q standard, and AV+Q safety with prod_debug_extra variant.

6.6 Inter-Process Communication

The NvScilpc library provides API for any two (2) entities in a system to communicate with each other irrespective of where they are placed. Entities can be:

- in different threads in the same process.
- in the same process.
- in different processes in the same VM.
- in different VMs on the same SoC.
- in different SoCs.

Each of these different boundaries are abstracted by a library that provides unified communication (read/write) API to entities.

Terminology

Channel: An NvScilpc channel connection allows bidirectional exchange of fixed-length messages between exactly two NvScilpc endpoints.

Endpoint: A software entity that uses NvScilpc channel to communicate with another software entity.

Frame: A message that consists of a sequence of bytes that is sent along an NvScilpc channel from one of the two NvScilpc endpoints of the NvScilpc channel to the other NvScilpc endpoint.

Frame Size: The size, in bytes, of every message exchanged along an NvScilpc channel. Each NvScilpc channel may have a distinct frame size.

Frame Count: The maximum number of NvScilpc frames that may simultaneously be queued for transfer in each direction along an NvScilpc channel.

Backend: An NvScilpc backend implements NvScilpc functionality for a particular class of NvScilpc channels. For example, for NvScilpc communication confined to SoCs, there are five different classes of NvScilpc channels depending on the maximum level of separation between NvScilpc endpoints that is supported by the NvScilpc channel.

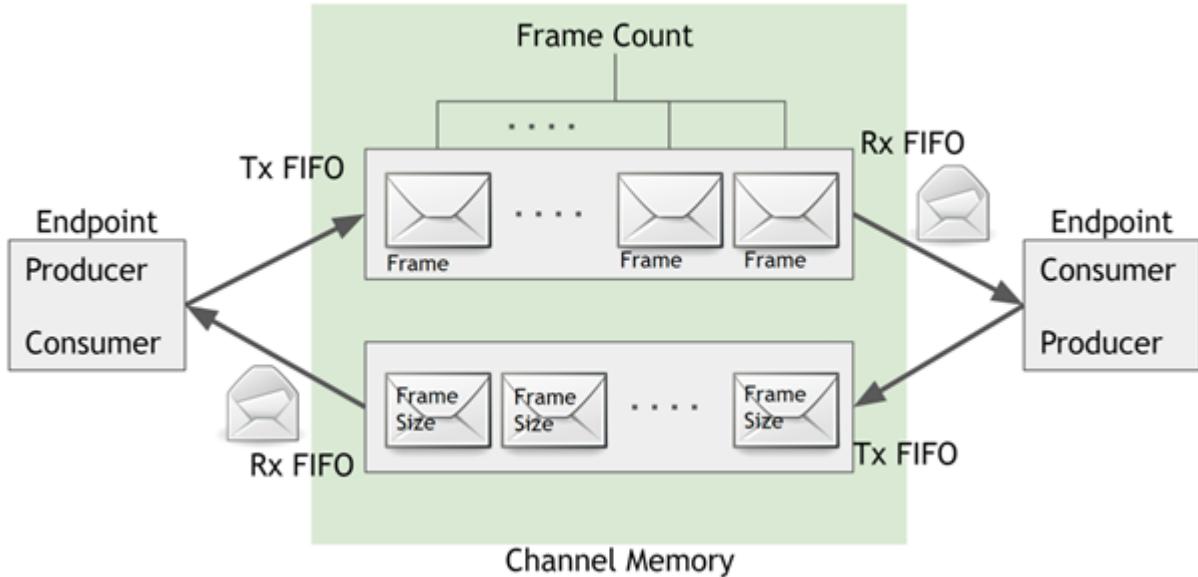
- **INTER_THREAD:** Handles communication between entities that may be in different threads in the same process.
- **INTER_PROCESS:** Handles communication between entities that may be in different processes in the same VM.
- **INTER_VM:** Handles communication between entities that may be in different VMs in the same SoC.
- **INTER_CHIP_PCIE:** Handles communication between entities that may be in different SoCs.

Endpoint Handle: Abstract data type that is passed to all NvScilpc channel communication.

Channel Reset: Defines the abrupt end of communication by one of the NvScilpc endpoints. In case of reset, no communication is allowed over NvScilpc channel until both endpoints reset their internal states and are ready for communication. NvScilpc relies on its backend to implement the channel reset mechanism. It may require cooperation from both endpoints where two endpoints have to wait for confirmation that the other has reset its local state.

Notification: An asynchronous signal along an NvScilpc channel through which one of the two endpoints of the NvScilpc channel indicates to the other NvScilpc endpoint that there may be an event for the latter NvScilpc endpoint to process.

6.6.6.1 NvScilpc Channel Memory



NvScilpc shared memory region is divided into two identical FIFOs: one FIFO on which the caller sends data, and one FIFO on which the caller receives data. The elements within FIFO are called Frame. Both FIFOs have an equal number (frame count) of frames, and all frames are a uniform fixed length (frame size).

Producer (or sender) constructs a message and sends it to FIFO. The FIFO queue keeps this message until the consumer is available. When a consumer is ready to process the message, it consumes the message from FIFO. The user can communicate remotely, as unidirectional point to point style, or bidirectional RPC (Remote Procedure Call) style (the producer sends a message and expects a response from the consumer).

The shared memory operation of NvScilpc is similar with message queue logically.

NvScilpc is a simple lightweight communication layer that can be used to send and receive raw data. NvScilpc doesn't provide error checking (such as CRC) or control field, which has frame length or frame type since they are overhead. The client is responsible for implementing or maintaining them.

Determining the Frame Size

Frame Size is the maximum data size which the producer (or sender) or consumer (or receiver) can send/receive with a single `NvSciIpcWriteSafe()`/`NvSciIpcReadSafe()` API call.

In considering cache coherency, frame size should be aligned with 64 bytes. There is no upper limit in frame size.

NvScilpc supports fixed-length frames, but if you need a variable-length frame protocol, set the max data size to the frame size and have the header (control field) and payload data structure for each frame. The header field might have frame length

or user frame type information. User can peek a specific header field using the `NvSciIpcReadGetNextFrame()` API before reading data using `NvSciIpcReadSafe()`.

Determining the Frame Count (= Queue Capacity)

The frame count depends on statistics (or usecases) of the user communication model. If the frame count is too small, an overflow/underflow issues may occur and messages may not be processed in time. This can cause unwanted flow control signaling and lead to performance degradation. If the frame count is too large, memory is wasted and processing efficiency is degraded. Choosing the right frame count is important for optimizing system performance (use the power of 2 as the frame count test value: 2, 4, 8, 16, 32, 64, 128, 256, 512 ...).

There is no upper limit with the frame count. You might need a higher frame count for buffering frames in case of streaming or burst mode communication.

For throughput-critical usecases, higher frame counts typically increase throughput, but once the frame count exceeds a certain value, throughput no longer increases.

Total Channel Memory Size

`Frame Size (in Bytes) x Frame Count x 2 (bidirectional FIFO)`

6.6.6.2 NvScilpc Configuration Data

NvScilpc configuration data is used to define all the channels present for a given Guest VM. Currently these details are provided via the plain text configuration file, where each line contains details about a single NvScilpc channel in the system.

Each channel entry is added to the plain text configuration file in string list form.

For INTER_THREAD and INTER_PROCESS backend, the format is :

```
<Backend-Name>, <Endpoint-Name>, <Endpoint-Name>, <Backend-Specific-Data>,
```

For INTER_VM and INTER_CHIP backend, the format is :

```
<Backend-Name>, <Endpoint-Name>, <Backend-Specific-Data>,
```

`<Endpoint-Name>` is unique string that is used to tag/identify a single NvScilpc endpoint in a system. Ideally it should describe the purpose for which the NvScilpc endpoint is created.

For INTER_THREAD and INTER_PROCESS backends, two endpoint names must be defined.

`<Backend-Name>` must be one of the following:

- INTER_THREAD
- INTER_PROCESS
- INTER_VM
- INTER_CHIP_PCIE

> INTER_CHIP

<Backend-Specific-Data> may span multiple fields.

The INTER_THREAD and INTER_PROCESS backend contains two (2) integer fields that describe <No-of-Frames Frame-Size> tuple. <Frame-Size> must be multiples of 64 bytes.

INTER_VM contains a single integer field that denotes the IVC queue ID.

INTER_CHIP_PCIE contains a SGID of C2C device node.

INTER_CHIP contains a single integer field that denotes the inter-chip device number.

6.6.6.2.1 Example NvScilpc config file format

```
# <Backend_name> <Endpoint-name1> <Endpoint-name2>
  <backend-specific-info>
INTER_PROCESS ipc_test_0    ipc_test_1    64    1536
INTER_PROCESS ipc_test_a_0   ipc_test_a_1   64    1536
INTER_PROCESS ipc_test_b_0   ipc_test_b_1   64    1536
INTER_PROCESS ipc_test_c_0   ipc_test_c_1   64    1536
INTER_THREAD  itc_test_0    itc_test_1    64    1536
INTER_VM      ivm_test      255
INTER_VM      loopback_tx   256
INTER_VM      loopback_rx   257
```

6.6.6.3 Adding a New Channel

6.6.6.3.1 Adding a New INTER_THREAD and INTER_PROCESS Channel

INTER_THREAD and INTER_PROCESS channels are implemented using POSIX shared memory, and POSIX mqueue for notifications. You must add a new line in the /etc/nvscilpc.cfg file describing the new channel.

Reboot the Linux VM if the added endpoint is for the secure memory use case; otherwise, there is no need to boot the Linux VM . Restart the NvScilpc application once nvscilpc.cfg is updated.

There is no specific limitation of Intra-VM channel capacity for now.

The inter-VM backend has 512 max channel entries for the entire NVIDIA DRIVE OS.

The inter-Chip backend has 16 max channel entries for the entire NVIDIA® DRIVE OS.

NvScilpc supports up to 500 endpoints per NvScilpc client process. (equal to 250 INTER_THREAD channels, or 500 INTER_PROCESS/INTER_VM channels)

6.6.6.3.2 Adding a New INTER_VM Channel

The INTER_VM channel relies on Hypervisor to set up the shared area details between two (2) VMs. At present, it is done via IVC queues that are described in the PCT. For any new INTER_VM channel:

1. Add a new IVC queue between two (2) VMs to the PCT file (`platform_config.h`) of the corresponding platform. The VM partition IDs are defined in the `server-partitions.mk` makefile. The `frame_size` value is in multiples of 64 bytes. The maximum IVC queue entries are 512 (the value imposed by the NVIDIA DRIVE® OS Hypervisor kernel). The location of the configuration file and makefile are as follows:
 - > `drive-foundation/platform-config/hardware/nvidia/platform/t23x/automotive/pct/drive_av/platform_config.h`
 - > `drive-foundation/virtualization/pct/make/t23x/server-partitions.mk`
2. When INTER_VM IVC notification latency is critical between different PCPUs, the user can choose the MSI-based (Message Signaled Interrupt) IVC notification by adding `use_msi = 1` option in the IVC queue table. The user should contact NVIDIA to use the MSI-based IVC notification since the total MSI-based IVC channel count is limited in the NVIDIA DRIVE OS® system. TRAP-based IVC notification is used by default if you don't specify the `use_msi` flag.
3. If the INTER_VM channel is defined in the configuration data of the cfg file but its IVC queue ID is NOT available in the PCT, that channel is ignored.
4. If the INTER_VM channel is defined in the configuration data of the cfg file but its IVC queue ID is NOT available in the PCT, that channel is ignored.

Example: IVC queue table format of PCT

```
.ivc = {
.queue = {
  ... skipped ...
  [queue id] = { .peers = {VM1_ID, VM2_ID}, .nframes = ##, .frame_size = ##, .use_msi
= # },
  ... skipped ...
},
... skipped ...
}

/* example */
[255] = { .peers = {GID_GUEST_VM, GID_UPDATE}, .nframes = 64, .frame_size = 1536 },
or

[255] = { .peers = {GID_GUEST_VM, GID_UPDATE}, .nframes = 64, .frame_size =
1536, .use_msi = 1 },
```

6.6.6.4 Update System Resources

NvScilpc must open endpoint-related resources such as shared memory, message queues, and device node. When the number of endpoints increases, the number of resources to open also increases, and it can exceed the system resource limit, which leads to NvScilpc failure at initialization.

The following two system resources are updated to support max endpoint count:

- > RLIMIT_NOFILE—maximum number of files to open
- > RLIMIT_MSGQUEUE—maximum number bytes to be allocated for message queue

The resources can be increased in `/etc/security/limits.d/driveos-limits.conf`.

```
$cat /etc/security/limits.d/driveos-limits.conf
* hard nofile 131072
* soft nofile 131072
root hard nofile 131072
root soft nofile 131072
* hard msgqueue 134217728
* soft msgqueue 134217728
root hard msgqueue 134217728
root soft msgqueue 134217728

#Check your current resource limits using "ulimit -a" command then compare it to above
settings.
#If your limits are lower than recommended values, you have to update them.
```



Note:

Existing sessions still have old limit values.

To apply new limits, reboot the system or use a new session (login).

`/etc/security/limits.d/driveos-limits.conf` overrides the settings in the `/etc/security/limits.conf` during boot time.

Regarding msgqueue, NvScilpc has one more tuning point in `/etc/sysctl.conf`.

```
$cat /etc/sysctl.conf
fs.mqueue.msg_max=256
fs.mqueue.queues_max=16384

#Check your current resource limits using "sysctl fs.mqueue.msg_max" and "sysctl
fs.mqueue.queues_max" command, and compare it to upper setting.
#If your limits are lower than recommended values, you have to update them.
```



Note:

To apply new limits, reboot the system or execute the `sudo sysctl -p` command.

6.6.6.5 NvScilpc API Usage

Each application first has to call `NvSciIpcInit()` before using any of the other NvScilpc APIs. This initializes the NvScilpc library instance for that application.



Note:

`NvSciIpcInit()` must be called by application only once at startup..

Initializing the NvScilpc Library

```
NvSciError err;
err = NvSciIpcInit();
if (err != NvSciError_Success) {
    return err;
}
```

6.6.6.5.1 Prepare an NvScilpc Endpoint for read/write

To enable read/write on an endpoint, the following steps must be completed.

1. The application must open the endpoint.
2. Get the endpoint information, such as the number of frames and each frame size. This is important as only single frames can be read/written at a given time.
3. Get the FD associated with the endpoint. This is required to handle event notifications.
4. Reset the endpoint. This is important as it ensures that the endpoint is not reading/writing any stale data (for example, from the previous start or instance).

Prepare an NvScilpc Endpoint for read/write

```
NvSciIpcEndpoint ipcEndpoint;
struct NvSciIpcEndpointInfo info;
int32_t fd;
NvSciError err;
err = NvSciIpcOpenEndpoint("ipc_endpoint", &ipcEndpoint);
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciIpcGetLinuxEventFd(ipcEndpoint, &fd);
if (err != NvSciError_Success) {
    goto fail;
}
err = NvSciIpcGetEndpointInfo(ipcEndpoint, &info);
if (err != NvSciError_Success) {
    goto fail;
}
printf("Endpointinfo: nframes = %d, frame_size = %d\n", info.nframes, info.frame_size);
err = NvSciIpcResetEndpointSafe(ipcEndpoint);
if (err != NvSciError_Success) {
    goto fail;
}
```

6.6.6.5.2 Writing to the NvScilpc Endpoint

The following example shows how to write to the NvScilpc endpoint.

Write to NvScilpc Channel

```
NvSciIpcEndpoint ipcEndpoint;
struct NvSciIpcEndpointInfo info;
int32_t fd;
fd_set rfds;
uint32_t event = 0;
```

```

void *buf;
uint32_t buf_size, bytes;
int retval;
NvSciError err;
FD_ZERO(&rfds);
FD_SET(fd, &rfds);
buf = malloc(info.frame_size);
if (buf == NULL) {
    goto fail;
}
while (1) {
    err = NvSciIpcGetEventSafe(ipcEndpoint, &event);
    if (err != NvSciError_Success) {
        goto fail;
    }
    if (event & NV_SCI_IPC_EVENT_WRITE) {
        /* Assuming buf contains the pointer to data to be written.
         * buf_size contains the size of data. It should be less than
         * Endpoint frame size.
         */
        err = NvSciIpcWrite(ipcEndpoint, buf, buf_size, &bytes);
        if (err != NvSciError_Success) {
            printf("error in writing endpoint\n");
            goto fail;
        }
    } else {
        retval = select(fd + 1, &rfds, NULL, NULL, NULL);
        if ((retval < 0) & (retval != EINTR)) {
            exit(-1);
        }
    }
}
}

```

6.6.6.5.3 Reading from the NvScilpc Endpoint

Read from NvScilpc channel

```

NvSciIpcEndpoint ipcEndpoint;
struct NvSciIpcEndpointInfo info;
int32_t fd;
fd_set rfds;
uint32_t event = 0;
void *buf;
uint32_t buf_size, bytes;
int retval;
NvSciError err;
FD_ZERO(&rfds);
FD_SET(fd, &rfds);
buf = malloc(info.frame_size);
if (buf == NULL) {
    goto fail;
}
while (1) {
    err = NvSciIpcGetEventSafe(ipcEndpoint, &event);

```

```

    if (err != NvSciError_Success) {
        goto fail;
    }
    if (event & NV_SCI_IPC_EVENT_READ) {
        /* Assuming buf contains pointer to area where frame is read.
         * buf_size contains the size of data. It should be less than
         * Endpoint frame size.
         */
        err = NvSciIpcReadSafe(ipcEndpoint, buf, buf_size, &bytes);
        if(err != NvSciError_Success) {
            printf("error in reading endpoint\n");
            goto fail;
        }
    } else {
        retval = select(fd + 1, &rfds, NULL, NULL, NULL);
        if ((retval < 0) & (retval != EINTR)) {
            exit(-1);
        }
    }
}

```

6.6.6.5.4 Cleaning-up an NvScilpc Endpoint

Once read/write is completed, you must free and clean the resources that were allocated by NvScilpc endpoints.

Clean up the Endpoint

```

NvSciIpcEndpoint ipcEndpoint;
bool clear = false;
err = NvSciIpcCloseEndpointSafe(ipcEndpoint, clear);
if (err != NvSciError_Success) {
    goto fail;
}

```

6.6.6.5.5 De-Initialize NvScilpc Library

```
NvSciIpcDeinit();
```

6.6.6.6 NvSciEventService API Usage

NvSciEventService is an event-driven framework that provides OS-agnostic APIs to send events and wait for events. The framework enables you to build portable event-driven applications and simplifies the steps required to prepare endpoint connections.

Initializing the NvSciEventService Library

Each application must call `NvSciEventLoopServiceCreateSafe()` before using any of the other `NvSciEventService` and `NvScilpc` APIs. This call initializes the `NvSciEventService` library instance for the application.


Note:

`NvSciEventLoopServiceCreateSafe()` must be called by the application only once at startup. Only single loop service is currently supported.

```
NvSciEventLoopService *eventLoopService;
NvSciError err;
err = NvSciEventLoopServiceCreateSafe(1, NULL, &eventLoopService);
if (err != NvSciError_Success) {
    goto fail;
}
err = NvScilpcInit();
if (err != NvSciError_Success) {
    return err;
```

Update System Resource

`NvScilpc` must open endpoint-related resources, such as shared memory, message queues, and device node. When the number of endpoints increases, the number resources to open also increases, which can exceed the system resource limit, leading to `NvScilpc` failure at initialization.

These two system resources matter:

- > `RLIMIT_NOFILE` – maximum number of files to open
- > `RLIMIT_MSGQUEUE` – maximum number bytes to be allocated for message queue

They can be increased in `/etc/security/limits.conf`.

```
$cat /etc/security/limits.conf
* hard nofile 32768
* soft nofile 32768
* hard msgqueue 18874368
* soft msgqueue 18874368
root hard msgqueue 18874368
root soft msgqueue 18874368

#Check your current resource limits using "ulimit -a" command then compare it to above
settings.
#If your limits are lower than recommended values, you have to update them.
$sudo sed -i '$ a * hard nofile 32768' /etc/security/limits.conf
$sudo sed -i '$ a * soft nofile 32768' /etc/security/limits.conf
$sudo sed -i '$ a * hard msgqueue 18874368' /etc/security/limits.conf
$sudo sed -i '$ a * soft msgqueue 18874368' /etc/security/limits.conf
$sudo sed -i '$ a root hard msgqueue 18874368' /etc/security/limits.conf
```

```
$sudo sed -i '$ a root soft msgqueue 18874368' /etc/security/limits.conf
```

**Note:**

Existing sessions retain old limit values.

To apply new limits, reboot the system or use a new session (login).

`/etc/security/limits.d/nofile.conf` overrides settings in the `/etc/security/limits.conf` during booting time.

6.6.6.6.1 Waiting for a Single Event for Read/Write

Before reading data, a connection must be established between two endpoint processes. This can be done by calling `NvSciIpcGetEventSafe()`.

Basic event handling mechanism is already described in [Writing to the NvSciIpc Endpoint](#) section. (The only difference is calling `WaitForEvent()` instead of `select()`).

**Note:**

`WaitForEvent()` is an event-blocking call. It must be called from a single thread only.

To process the write event in a loop, add the `WRITE` event check routine and `NvSciIpcWriteSafe()`.

Wait for a single event for read

```
NvSciEventLoopService *eventLoopService;
NvSciIpcEndpoint ipcEndpoint;
NvSciEventNotifier *eventNotifier;
struct NvSciIpcEndpointInfo info;
int64_t timeout;
uint32_t event = 0;
void *buf;
int32_t buf_size, bytes;
int retval;
NvSciError err;
timeout = NV_SCI_EVENT_INFINITE_WAIT;
buf = malloc(info.frame_size);
if (buf == NULL) {
    goto fail;
}
buf_size = info.frame_size;
while (1) {
    err = NvSciIpcGetEventSafe(ipcEndpoint, &event);
    if (err != NvSciError_Success) {
        goto fail;
    }
    if (event & NV_SCI_IPC_EVENT_READ) {
        /* Assuming buf contains pointer to area where frame is read.
         * buf_size contains the size of data. It should be less than
```

```

        * Endpoint frame size.
        */
err = NvSciIpcReadSafe(ipcEndpoint, buf, buf_size, &bytes);
if(err != NvSciError_Success) {
    printf("error in reading endpoint\n");
    goto fail;
}
} else {
    err = eventLoopService->WaitForEvent(eventNotifier, timeout);
if(err != NvSciError_Success) {
    printf("error in waiting event\n");
    goto fail;
}
}
}
}

```

6.6.6.6.2 Waiting for Multiple Events for Read/Write

In this scenario, multiple endpoints are opened, and multiple event notifiers are created

A connection must be established between two endpoint processes before reading the data. This can be done by calling `NvSciIpcGetEventSafe()`.

The event handling mechanism is similar to waiting for a single event, but it can wait for multiple events.

Check the `newEventArray` boolean array returned by `WaitForMultipleEvents()` or `WaitForMultipleEventsExt()` to determine which event is notified.

`WaitForMultipleEventsExt()` has the same functionality as `WaitForMultipleEvents()` and extends to be used in case no notifier array is available but events want to be serviced with the event handler when the event arrives.



Note:

`WaitForEvent()` is an event-blocking call. It must be called from a single thread only.

To process the write event in a loop, add WRITE event check routine and `NvSciIpcWriteSafe()`.

Following is a list of supported events:

- > Multiple native events
- > Multiple local events
- > Multiple native and local events

Wait for multiple events for read/write

```

#define NUM_OF_EVENTNOTIFIER 2
NvSciEventLoopService *eventLoopService;
NvSciIpcEndpoint ipcEndpointArray[NUM_OF_EVENTNOTIFIER];
NvSciEventNotifier *eventNotifierArray[NUM_OF_EVENTNOTIFIER];
bool newEventArray[NUM_OF_EVENTNOTIFIER];

```

```

struct NvSciIpcEndpointInfo infoArray; /* Assuming two endpoints have the same info */
int64_t timeout;
uint32_t event = 0;
void *buf;
int32_t buf_size, bytes, inx;
int retval;
NvSciError err;
bool gotEvent;
timeout = NV_SCI_EVENT_INFINITE_WAIT;
buf = malloc(info.frame_size);
if (buf == NULL) {
    goto fail;
}
buf_size = info.frame_size;
for (inx = 0; inx < NUM_OF_EVENTNOTIFIER; inx++) {
    newEventArray[inx] = true;
}
while (1) {
    gotEvent = false;
    for (inx = 0; inx < NUM_OF_EVENTNOTIFIER; inx++) {
        if (newEventArray[inx]) {
            err = NvSciIpcGetEventSafe(ipcEndpointArray[inx], &event);
            if (err != NvSciError_Success) {
                goto fail;
            }
            if (event & NV_SCI_IPC_EVENT_READ) {
                /* Assuming buf contains pointer to area where frame is read.
                 * buf_size contains the size of data. It should be less than
                 * Endpoint frame size.
                 */
                err = NvSciIpcReadSafe(ipcEndpointArray[inx], buf, buf_size, &bytes);
                if (err != NvSciError_Success) {
                    printf("error in reading endpoint\n");
                    goto fail;
                }
                gotEvent = true;
            }
        }
    }
    if (gotEvent) {
        continue;
    }
    err = eventLoopService->WaitForMultipleEvents(eventNotifierArray,
                                                NUM_OF_EVENTNOTIFIER, timeout, newEventArray);
    if (err != NvSciError_Success) {
        printf("error in waiting event\n");
        goto fail;
    }
}

```

6.6.6.6.3 Creating a Local Event

A local event does not require an associated endpoint. It uses two threads of a process. When creating a local event, use `NvSciEventLoopServiceCreateSafe()` instead of `NvSciIpcInit`.

One thread called a sender is for sending a signal and the other called receiver is for waiting for the signal.

The application must first create a local event by calling `EventService.CreateLocalEvent()`. This call also creates an event notifier and associates the notifier with the local event.

Create local event

```
NvSciEventLoopService *eventLoopService;
NvSciLocalEvent *localEvent;
NvSciError err;
err = eventLoopService->EventService.CreateLocalEvent(
    &eventLoopService->EventService,
    &localEvent);
if (err != NvSciError_Success) {
    goto fail;
}
```

6.6.6.6.4 Sending a Signal with a Local Event

A sender in the same or different thread can send a signal to a receiver by calling `Signal()`.

Send a signal with local event

```
NvSciLocalEvent *localEvent;
NvSciError err;
err = localEvent->Signal(localEvent);
if (err != NvSciError_Success) {
    goto fail;
}
```

6.6.6.6.5 Waiting for a Local Event

A sender can send a signal to a receiver in the same or different thread.

A receiver in the same or a different thread can be notified of the signal from sender.

The receiver uses `WaitForEvent` for a single signal or `WaitForMultipleEvents`/`WaitForMultipleEventsSafe` for multiple signals or mixed events associated with an endpoint.



Note:

`WaitForEvent()` is an event-blocking call. It must be called from a single thread only.

Here is a list of supported events:

- Single native event
- Single local events

Wait for a Local Event

```
NvSciEventLoopService *eventLoopService;
NvSciLocalEvent *localEvent;
NvSciError err;
int64_t timeout;
timeout = NV_SCI_EVENT_INFINITE_WAIT;
while (1) {
    err = eventLoopService->WaitForEvent(localEvent->eventNotifier, timeout);
    if(err != NvSciError_Success) {
        printf("error in waiting event\n");
        goto fail;
    }
}
/* Do something with the local event notified */
}
```

6.6.6.6.6 Cleaning Up Event Notifier and Local Event

Event notifiers for local events and for native events are deleted when they are no longer used. A local event is deleted explicitly by an application while a native event is deleted implicitly by an event notifier.

When deleting events, the application must delete the event notifier before the local event.

Clean up an event notifier and a local event

```
NvSciEventNotifier *nativeEventNotifier; /* Assume
this is event notifier for native event */
NvSciLocalEvent *localEvent;
nativeEventNotifier->Delete(nativeEventNotifier);
local->eventNotifier->Delete(local->eventNotifier);
local->Delete(local);
```

6.6.6.7 De-Initializing NvSciEventService Library

The application must call `EventService.Delete()` after de-initializing the NvSciIpc library.

De-initialize NvSciEventService library

```
NvSciIpcEndpoint ipcEndpoint;
NvSciEventLoopService *eventLoopService;

NvSciIpcCloseEndpoint(ipcEndpoint);
NvSciIpcDeinit();

eventLoopService->EventService.Delete
(&eventLoopService->EventService);
```

6.6.7 Chip to Chip Communication

The NVIDIA® Software Communication Interface for Chip to Chip over direct PCIe connection (NvSciC2cPcie) provides the ability for user applications to exchange data across two NVIDIA DRIVE AGX Orin™ DevKits interconnected on a direct PCIe connection. The direct PCIe connection is between the first/one NVIDIA DRIVE AGX Orin Developer Kits as a PCIe Root Port with the second/other NVIDIA DRIVE AGX Orin DevKit as a PCIe Endpoint.

Supported Platform Configurations

Platform

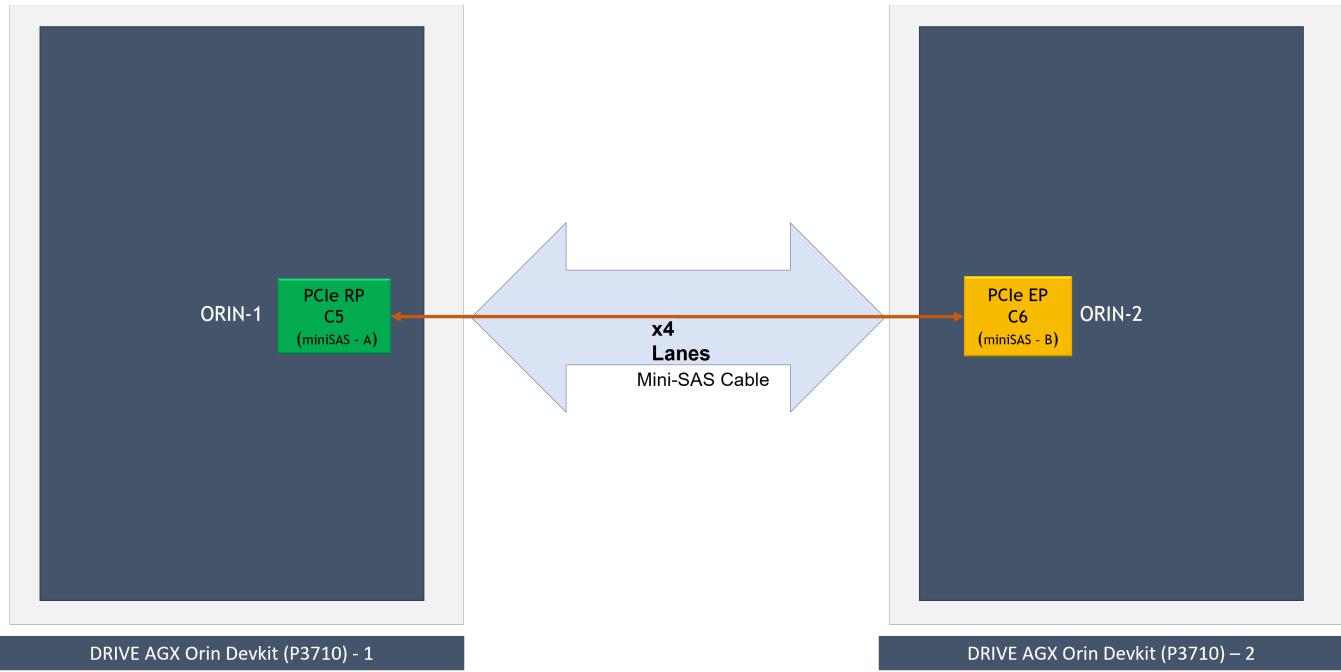
- > NVIDIA DRIVE® AGX Orin DevKit
- > NVIDIA DRIVE Recorder

SoC

- > NVIDIA DRIVE Orin as PCIe Root Port
- > NVIDIA DRIVE Orin as PCIe Endpoint

Topology

- > NVIDIA DRIVE AGX Orin DevKit as PCIe Root Port <> NVIDIA DRIVE AGX Orin DevKit as PCIe Endpoint



- > NVIDIA DRIVE Recorder Orin A as PCIe Endpoint <> NVIDIA DRIVE Recorder Orin B as Root Port

Platform Setup

The following platform configurations are required for NvSciC2cPcie communication with NVIDIA DRIVE AGX Orin DevKit. Similar connections are required for other platforms.

- > miniSAS Port-A of NVIDIA DRIVE AGX Orin DevKit - 1 connected to miniSAS Port-B of NVIDIA DRIVE AGX Orin DevKit - 2 with a PCIe miniSAS cable.
- > The PCIe controllers of the two NVIDIA DRIVE AGX Orin DevKits when interconnected back-to-back have PCIe re-timers, and the PCIe re-timer firmware must be flashed for the appropriate PCIe lane configuration.
- > For custom platform PCIe controllers used, configure lane and clock accordingly.
- > Each PCIe controller in NVIDIA DRIVE AGX Orin Devkit has PCIe EDMA engine. NvSciC2cPcie uses only one DMA Write channel of the assigned PCIe controller for all the NvSciC2cPcie transfers.
 - NvSciC2cPcie transfer is in the FIFO mechanism, and there is no load balancing or scheduling policy to prioritize the specific request.

Execution Setup

Linux Kernel Module Insertion

NvSciC2cPcie only runs on select platforms: NVIDIA DRIVE AGX Orin DevKit and NVIDIA DRIVE Recorder. Before user applications can exercise NvSciC2cPcie interface, you must insert the Linux kernel modules for NvSciC2cPcie. They are not loaded by default on NVIDIA DRIVE® OS Linux boot. To insert the required Linux kernel module:

- > On first/one Orin configured as PCIe Root Port

```
sudo modprobe nvscic2c-pcie-epc
```

- > On second/other Orin DevKit configured as PCIe Endpoint

```
sudo modprobe nvscic2c-pcie-epf
```

A recommendation is to load nvscic2c-pcie-ep* kernel modules immediately after boot. This allows the nvscic2c-pcie software stack to allocate contiguous physical pages for its internal operation for each of the nvscic2c-pcie endpoints configured.

PCIe Hot-Plug

Once loaded, Orin DevKit enabled as PCIe Endpoint is hot-plugged and enumerated as a PCIe device with Orin DevKit configured as PCIe Root Port (miniSAS cable connected to miniSAS port-A). The following must be executed on Orin DevKit configured as PCIe Endpoint (miniSAS cable connected to miniSAS port-B):

```
sudo -s
cd /sys/kernel/config/pci_ep/
mkdir functions/nvscic2c_epf_22CC/func
echo 0x10DE > functions/nvscic2c_epf_22CC/func/vendorid
echo 0x22CC > functions/nvscic2c_epf_22CC/func/deviceid
ln -s functions/nvscic2c_epf_22CC/func controllers/141c0000.pcie_ep
echo 0 > controllers/141c0000.pcie_ep/start
echo 1 > controllers/141c0000.pcie_ep/start
```

The previous steps, including Linux kernel module insertion, can be added as a `linux` `systemd` service to facilitate auto-availability of NvSciC2cPcie software at boot.

NvScilpc (INTER_CHIP, PCIe) Channels

Once the Linux kernel module insertion and PCIe hot-plug completes successfully, the following NvScilpc channels are available for use with NvStreams producer or consumer applications.

NVIDIA DRIVE AGX Orin DevKit

NVIDIA DRIVE AGX Orin DevKit as PCIe Root Port	NVIDIA DRIVE AGX Orin DevKit as PCIe Endpoint
<code>nvscic2c_PCIE_s0_c5_1</code>	<code>nvscic2c_PCIE_s0_c6_1</code>
<code>nvscic2c_PCIE_s0_c5_2</code>	<code>nvscic2c_PCIE_s0_c6_2</code>
<code>nvscic2c_PCIE_s0_c5_3</code>	<code>nvscic2c_PCIE_s0_c6_3</code>
<code>nvscic2c_PCIE_s0_c5_4</code>	<code>nvscic2c_PCIE_s0_c6_4</code>
<code>nvscic2c_PCIE_s0_c5_5</code>	<code>nvscic2c_PCIE_s0_c6_5</code>
<code>nvscic2c_PCIE_s0_c5_6</code>	<code>nvscic2c_PCIE_s0_c6_6</code>
<code>nvscic2c_PCIE_s0_c5_7</code>	<code>nvscic2c_PCIE_s0_c6_7</code>
<code>nvscic2c_PCIE_s0_c5_8</code>	<code>nvscic2c_PCIE_s0_c6_8</code>
<code>nvscic2c_PCIE_s0_c5_9</code>	<code>nvscic2c_PCIE_s0_c6_9</code>
<code>nvscic2c_PCIE_s0_c5_10</code>	<code>nvscic2c_PCIE_s0_c6_10</code>
<code>nvscic2c_PCIE_s0_c5_11</code>	<code>nvscic2c_PCIE_s0_c6_11</code>

`nvscic2c_PCIE_s0_c5_12` and `nvscic2c_PCIE_s0_c6_12` are not available for use with NvStreams over Chip to Chip connection but only for NvScilpc over Chip to Chip (INTER_CHIP, PCIe) specifically for short and less-frequent generic-purpose data. For additional information, see [NvScilpc API Usage](#)

NVIDIA DRIVE Recorder

NVIDIA DRIVE AGX Orin DevKit as PCIe Root Port	NVIDIA DRIVE AGX Orin DevKit as PCIe Endpoint
<code>nvscic2c_PCIE_s2_c6_1</code>	<code>nvscic2c_PCIE_s1_c6_1</code>
<code>nvscic2c_PCIE_s2_c6_2</code>	<code>nvscic2c_PCIE_s1_c6_2</code>
<code>nvscic2c_PCIE_s2_c6_3</code>	<code>nvscic2c_PCIE_s1_c6_3</code>

NVIDIA DRIVE AGX Orin DevKit as PCIe Root Port	NVIDIA DRIVE AGX Orin DevKit as PCIe Endpoint
nvscic2c_PCIE_s2_c6_4	nvscic2c_PCIE_s1_c6_4
nvscic2c_PCIE_s2_c6_5	nvscic2c_PCIE_s1_c6_5
nvscic2c_PCIE_s2_c6_6	nvscic2c_PCIE_s1_c6_6
nvscic2c_PCIE_s2_c6_7	nvscic2c_PCIE_s1_c6_7
nvscic2c_PCIE_s2_c6_8	nvscic2c_PCIE_s1_c6_8
nvscic2c_PCIE_s2_c6_9	nvscic2c_PCIE_s1_c6_9
nvscic2c_PCIE_s2_c6_10	nvscic2c_PCIE_s1_c6_10
nvscic2c_PCIE_s2_c6_11	nvscic2c_PCIE_s1_c6_11

NvScilpc (INTER_CHIP, PCIe) channel names can be modified according to convenience.

The user application on NVIDIA DRIVE AGX Orin DevKit as PCIe Root Port opens nvscic2c_PCIE_s0_c5_1 for use, then the peer user application on the other NVIDIA DRIVE AGX Orin DevKit as PCIe Endpoint must open nvscic2c_PCIE_s0_c6_1 for exchange of data across the SoCs and for the remaining channels listed previously. Similarly, for other platforms, endpoints must open respective channels as per the previous channels table.

Each of the NvScilpc (INTER_CHIP, PCIe) channels are configured to have 16 frames with 32 KB per frame as the default.

Reconfiguration

The following reconfiguration information is based on the default NvSciC2cPcie support offered for NVIDIA DRIVE AGX Orin DevKit.

Different platforms or a different PCIe controller configuration on the same NVIDIA DRIVE AGX Orin DevKit requires adding a new set of device-tree node entries for NvSciC2cPcie on a PCIe Root Port (`nvidia, tegra-nvscic2c-pcie-epc`) and a PCIe Endpoint (`nvidia, tegra-nvscic2c-pcie-epf`). For example, a change in PCIe Controller Id or a change in the role of a PCIe controller from PCIe Root Port to PCIe Endpoint (or vice versa) from the default NVIDIA DRIVE AGX Orin DevKit requires changes. The changes are possible with device-tree node changes or additions, but it is not straightforward to document them all. These are one-time changes and can occur in coordination with your NVIDIA point-of-contact.

BAR Size

BAR size for NVIDIA DRIVE AGX Orin as PCIe Endpoint is configured to 1 GB by default. When required, this can be reduced or increased by modifying the property `nvidia, bar-win-size` of device-tree node: `nvscic2c-pcie-s0-c6-epf`

File: <PDK_TOP>/drive-linux/kernel/source/hardware/nvidia/platform/t23x/automotive/kernel-dts/p3710/common/tegra234-p3710-0010-nvscic2c-pcie.dtsi

```
nvscic2c-pcie-s0-c6-epf {
    compatible = "nvidia,tegra-nvscic2c-pcie-epf";
--    nvidia,bar-win-size = <0x40000000>; /* 1GB. */
++    nvidia,bar-win-size = <0x20000000>; /* 512MB. */
};
```

The configured BAR size must be a power-of 2 and a minimum of 64 MB. Maximum size of BAR depends on the size of pre-fetchable memory supported by PCIe RP. With NVIDIA DRIVE AGX, max BAR size should be 0xf000 less than maximum pre-fetchable memory.

NvScilpc (INTER_CHIP, PCIe) Channel Properties

The NvScilpc (INTER_CHIP, PCIe) channel properties can be modified on a use-case basis.

Modify Channel Properties

To change the channel properties *frames* and *frame size*, must change for both NVIDIA DRIVE AGX Orin DevKit as PCIe Root Port and NVIDIA DRIVE AGX Orin DevKit as PCIe Endpoint device-tree nodes: nvscic2c-pcie-s0-c5-epc and nvscic2c-pcie-s0-c6-epf respectively.

file: <PDK_TOP>/drive-linux/kernel/source/hardware/nvidia/platform/t23x/automotive/kernel-dts/p3710/common/tegra234-p3710-0010-nvscic2c-pcie.dtsi

The following illustrates change in frames count or number of frames for NvScilpc (INTER_CHIP, PCIe) channel: nvscic2c_pcie_s0_c5_2 (PCIe Root Port) and nvscic2c_pcie_s0_c6_1(PCIe Endpoint)

```
nvscic2c-pcie-s0-c5-epc {
    nvidia,endpoint-db =
    "nvscic2c_pcie_s0_c5_1,      16,      00032768",
--    "nvscic2c_pcie_s0_c5_2,      16,      00032768",
++    "nvscic2c_pcie_s0_c5_2,      08,      00032768",
    "nvscic2c_pcie_s0_c5_3,      16,      00032768",
    ...
};

nvscic2c-pcie-s0-c6-epf {
    nvidia,endpoint-db =
    "nvscic2c_pcie_s0_c6_1,      16,      00032768",
--    "nvscic2c_pcie_s0_c6_2,      16,      00032768",
++    "nvscic2c_pcie_s0_c6_2,      08,      00032768",
    "nvscic2c_pcie_s0_c6_3,      16,      00032768",
    ...
};
```

The following illustrates change in frame size for NvScilpc (INTER_CHIP, PCIe) channel: nvscic2c_pcie_s0_c5_2 (PCIe Root Port) and nvscic2c_pcie_s0_c6_1(PCIe Endpoint)

```
nvscic2c-pcie-s0-c5-epc {
    nvidia,endpoint-db =
    "nvscic2c_pcie_s0_c5_1,      16,      00032768",
```

```
--  "nvscic2c_pcie_s0_c5_2,      16,      00032768",
++ "nvscic2c_pcie_s0_c5_2,      16,      00028672",
"nvscic2c_pcie_s0_c5_3,      16,      00032768",
...
};

nvscic2c-pcie-s0-c6-epf {
nvidia,endpoint-db =
"nvscic2c_pcie_s0_c6_1,      16,      00032768",
--  "nvscic2c_pcie_s0_c6_2,      16,      00032768",
++ "nvscic2c_pcie_s0_c6_2,      16,      00028672",
"nvscic2c_pcie_s0_c6_3,      16,      00032768",
...
};

}
```

New Channel Addition

To introduce additional NvScilpc (INTER_CHIP, PCIe) channels, the change must occur for both NVIDIA DRIVE AGX Orin DevKit as PCIe Root Port and NVIDIA DRIVE AGX Orin DevKit as PCIe Endpoint device-tree nodes: nvscic2c-pcie-s0-c5-epc and nvscic2c-pcie-s0-c6-epf respectively.

File: <PDK_TOP>/drive-linux/kernel/source/hardware/nvidia/platform/t23x/automotive/kernel-dts/p3710/common/tegra234-p3710-0010-nvscic2c-pcie.dtsi

```
nvscic2c-pcie-s0-c5-epc {
nvidia,endpoint-db =
"nvscic2c_pcie_s0_c5_1,      16,      00032768",
...
--  "nvscic2c_pcie_s0_c5_11,      16,      00032768";
++ "nvscic2c_pcie_s0_c5_11,      16,      00032768",
++ "nvscic2c_pcie_s0_c5_12,      16,      00032768";
};

nvscic2c-pcie-s0-c6-epf {
nvidia,endpoint-db =
"nvscic2c_pcie_s0_c6_1,      16,      00032768",
...
--  "nvscic2c_pcie_s0_c6_11,      16,      00032768";
++ "nvscic2c_pcie_s0_c6_11,      16,      00032768",
++ "nvscic2c_pcie_s0_c6_12,      16,      00032768";
};
```

File: /etc/nvsciipc.cfg(on target)

```
INTER_CHIP_PCIE      nvscic2c_pcie_s0_c5_11  0000
++ INTER_CHIP_PCIE      nvscic2c_pcie_s0_c5_12  0000
...
...
...
INTER_CHIP_PCIE      nvscic2c_pcie_s0_c6_11  0000
++ INTER_CHIP_PCIE      nvscic2c_pcie_s0_c6_12  0000
```

Changes can be made to reduce, subtract, or remove any of the existing NvScilpc (INTER_CHIP, PCIe) channels.

For a given pair of NVIDIA DRIVE AGX Orin DevKit as PCIe Root Port and NVIDIA DRIVE AGX Orin DevKit as PCIe Endpoint, the maximum NvScilpc (INTER_CHIP, PCIe) channels supported are 16.

Similarly for other platforms, corresponding *dtsi files require modification.

For NVIDIA DRIVE Recorder: <PDK_TOP>/drive-linux/kernel/source/hardware/nvidia/platform/t23x/automotive/kernel-dts/p4024/common/tegra234-p4024-nvscic2c-pcie.dtsi



Note: The channel properties have the following limits:

- > Frame count: minimum: 1, maximum: 64
- > Frame size: minimum: 64B, maximum: 32 KB. Must always be aligned to 64B

PCIe Hot-Unplug

To tear down the connection between PCIe Root Port and PCIe Endpoint, PCIe hot-unplug PCIe Endpoint from PCIe Root Port. Refer to the [Restrictions](#) section for more information.

The PCIe Hot-Unplug is always executed from PCIe Endpoint [NVIDIA DRIVE AGX Orin DevKit (miniSAS cable connected to miniSAS port-B)] by initiating the power-down off the PCIe Endpoint controller and subsequently unbinding the nvscic2c-pcie-epf module with the PCIe Endpoint.

Prerequisite: PCIe Hot-Unplug must be attempted only when the PCIe Endpoint is successfully hot-plugged into PCIe Root Port and NvScilpc(INTER_CHIP, PCIE) channels are enumerated.

To PCIe hot-unplug, execute the following on NVIDIA DRIVE AGX Orin DevKit configured as PCIe Endpoint (miniSAS cable connected to miniSAS port-B). This makes NvScilpc(INTER_CHIP, PCIE) channels disappear on both the PCIe inter-connected NVIDIA DRIVE AGX Orin DevKits.

```
sudo -s
cd /sys/kernel/config/pci_ep/
```

Check NvSciC2cPcie device nodes are available:

```
ls /dev/nvscic2c_*
```

The previous command should list NvSciC2cPcie device nodes for the corresponding PCIe Root Port and PCIe Endpoint connection. Continue with the following set of commands:

```
echo 0 > controllers/141c0000.pcie_ep/start
```

Wait until NvSciC2cPcie device nodes disappear. The following command can be used to check for NvSciC2cPcie device nodes availability:

```
ls /dev/nvscic2c_*
```

Once the device nodes disappear, execute:

```
unlink controllers/141c0000.pcie_ep/func
```

Successful PCIe hot-unplug of PCIe Endpoint from PCIe Root Port makes the NvScilpc(INTER_CHIP, PCIE) channels as listed, NvScilpc (INTER_CHIP, PCIe) channels, go away on both the NVIDIA DRIVE AGX Orin DevKits, and you can proceed with power-cycle/off of one or both the NVIDIA DRIVE AGX Orin DevKits.

PCIe Hot-Replug

To re-establish the PCIe connection between PCIe Endpoint and PCIe Root Port, you must PCIe hot-replug PCIe Endpoint to PCIe Root Port.

When both the SoCs are power-cycled after PCIe hot-unplug previously, you must follow the usual steps of PCIe hot-plug. However, if one of the two SoCs power-cycled/rebooted, then PCIe hot-replug is required to re-establish the connection between them.



Note: NVIDIA DRIVE AGX Orin DevKits with PCIe retimer firmware(FW) have known issues that do not allow users to PCIe hot-replug and PCIe hot-unplug without rebooting both the PCIe inter-connected NVIDIA DRIVE AGX Orin DevKits. For the platforms that have PCIe retimers, you must power-cycle/reboot both the NVIDIA DRIVE AGX Orin DevKits and establish the PCIe connection between them after PCIe hot-unplug. See the *Execution Setup* section for connection information.



Note: PCIe Hot-Unplug and Replug functionality has not been verified on NVIDIA DRIVE Recorder.

Prerequisite: PCIe hot-replug is attempted when one of the two SoCs is power-recycled/rebooted after a successful attempt of PCIe hot-unplug between them. If both SoCs were power-recycled/rebooted, then the same steps as listed in the *Execution Setup* section are required to establish the PCIe connection between them.

For platforms that do not have PCIe retimers, to achieve PCIe hot-replug after the connection was PCIe Hot-Unplugged before NVIDIA DRIVE AGX Orin DevKit has rebooted, execute the following on NVIDIA DRIVE AGX Orin DevKit configured as PCIe Endpoint (miniSAS cable connected to miniSAS port-B). This makes NvScilpc(INTER_CHIP, PCIE) endpoints reappear on both the PCIe inter-connected NVIDIA DRIVE AGX Orin DevKits.

When only PCIe Root Port SoC was power-recycled/rebooted

On PCIe Root Port SoC (NVIDIA DRIVE AGX Orin DevKit (miniSAS cable connected to miniSAS port-A))

Follow the same steps as listed in *Linux Kernel Module Insertion, Execution Setup*.

On PCIe Endpoint SoC [NVIDIA DRIVE AGX Orin DevKit(miniSAS cable connected to miniSAS port-B)]

```
sudo -s
cd /sys/kernel/config/pci_ep/
ln -s functions/nvscic2c_epf_22CC/func controllers/141c0000.pcie_ep
```

```
echo 0 > controllers/141c0000.pcie_ep/start
echo 1 > controllers/141c0000.pcie_ep/start
```

When only PCIe Endpoint SoC is power-recycled/rebooted

On PCIe Endpoint SoC (NVIDIA DRIVE AGX Orin DevKit (miniSAS cable connected to miniSAS port-B)

Follow the steps *Execution Setup*.

On PCIe Root Port SoC (NVIDIA DRIVE AGX Orin DevKit (miniSAS cable connected to miniSAS port-A)

Nothing is required. The module is already inserted.

SoC Error

The only scenario for SoC Error is when one or both of the PCIe Root Port SoC and PCIe Endpoint SoC connected with nvscic2c-pcie has linux-kernel oops/panic. The application might observe timeouts.

Reconnection

On the SoC that is still functional and responsive, user must follow the same restrictions for PCIe Hot-Unplug. On the same SoC, once the applications exit or pipeline is purged, user must recover the faulty SoC either by rebooting or resetting it.

Only then, subsequently:

- > If the functional SoC (non-faulty) was PCIe Endpoint SoC, then same steps as for 'PCIe Hot-Unplug' and 'PCIe Hot-Replug' listed above on PCIe Endpoint SoC and on the recovered SoC (PCIe Root Port SoC) user must do the same steps as listed in sub-section 'Linux Kernel Module Insertion'
- > If the functional SoC (non-faulty) was PCIe Root Port SoC, then nothing is to be done on that SoC, but on the recovered SoC (PCIe Endpoint SoC) user must do the same steps as listed in the PCIe Hot-Plug section.
- > If both the SoC's were faulty, then on recovering each of the two SoCs, it becomes the usual case of 'Linux kernel Module Insertion' and 'PCIe Hot-Plug' as done to establish the PCIe connection between them initially.

Successful Error Recovery and PCIe reconnection makes the Channels reappear/available again for use.

PCIe Error

Case 1: PCIe Link Errors (AER)

PCIe AER resulting on either PCIe Root Port SoC or PCIe Endpoint SoC are always reported on PCIe Root Port. Only PCIe Uncorrectable AER(s) are reported.

Recovery

- > Perform steps for 'PCIe Hot-Unplug' and then 'PCIe Hot-Replug' listed above. Successful PCIe reconnection (PCIe Hot-Unplug/Replug) makes the Channels reappear/available again for use.

Case 2: PCIe EDMA Transfer Errors

PCIe EDMA transfer errors can lead to data loss.

Recovery

- > PCIe EDMA engine is sanitized once all pipelined transfers are returned. Recovery from PCIe EDMA errors is not guaranteed and therefore it is recommended to retry streaming and if error persists PCIe link recovery would be required.

For PCIe link recovery, 'PCIe Hot-Unplug' and then 'PCIe Hot-Replug' is required, as listed above on PCIe Endpoint SoC.

SC-7 Suspend and Resume Cycle

Follow the same set of restrictions and assumptions for SC-7 suspend and resume cycle as listed in the PCIe Hot-Unplug and PCIe Hot-Replug sections. Before one or both the two interconnected SoCs enter SC-7 suspend, PCIe Hot-Unplug must be carried out keeping the set of restrictions applicable for PCIe Hot-Unplug. Once one or both the two interconnected SoCs exit from SC-7 suspend, such as SC-7 resume, the same steps as listed in PCIe Hot-Replug are required.

Assumptions

- > NVIDIA Software Communication Interface for Chip to Chip (NvSciC2cPcie) is offered only between the inter-connected NVIDIA DRIVE Orin SoC as PCIe Root Port and a NVIDIA DRIVE Orin SoC as PCIe Endpoint. Producer buffers are copied onto remote consumer buffers pinned to PCIe memory using the PCIe eDMA engine.
- > NVIDIA Software Communication Interface for Chip to Chip (NvSciC2cPcie) is offered from a single Guest OS Virtual Machine of a NVIDIA DRIVE Orin SoC as PCIe Root Port to a single Guest OS Virtual Machine of another NVIDIA DRIVE Orin SoC as PCIe Endpoint.
- > User-applications are responsible for the steps to teardown the ongoing Chip to Chip transfer pipeline on all the SoCs in synergy and gracefully.
- > Out of the box support is ensured for NVIDIA DRIVE AGX Orin DevKit inter-connected with another NVIDIA DRIVE AGX Orin DevKit. In this configuration, the default configuration is PCIe controller C5 in PCIe Root Port mode and PCIe controller C6 in PCIe Endpoint mode. Any change in PCIe controller mode or by moving to another set of PCIe controllers for NvSciC2cPcie requires changes in the `tegra234-p3710-0010-nvscic2c-pcie.dtsi` device-tree include file.

Restrictions

- > Before powering-off/recycling one of the two PCIe inter-connected NVIDIA DRIVE AGX Orin DevKits when one NVIDIA DRIVE AGX Orin DevKit is PCIe hot-plugged into another NVIDIA DRIVE AGX Orin DevKit, you must tear down the PCIe connection between them (PCIe hot-unplug).
- > Before tearing down the PCIe connection between the two SoCs (PCIe hot-unplug), on both of these SoCs, all applications or streaming pipelines using the corresponding NvSciIpc(INTER_CHIP, PCIE) channels will exit or purge. Before they exit or purge, the corresponding in-use NvSciIpc(INTER_CHIP, PCIE) channel must be closed with `NvSciIpcCloseEndpointSafe()`.

- > On the two PCIe inter-connected NVIDIA DRIVE AGX Orin DevKits, before closing a corresponding NvScilpc(INTER_CHIP, PCIE) channel with NvSciIpcCloseEndpointSafe(), you must ensure for this NvScilpc(INTER_CHIP, PCIE) channel:
 - No pipelined NvSciSync waits are pending.
 - All the NvScilpc (INTER_CHIP, PCIE) channel messages sent have been received.
 - All the NvSciBuf and NvSciSync, source and target handles, export and import handles, registered and CPU mapped, with NvSciC2cPcie layer must be unregistered and their mapping deleted with NvSciC2cPcie layer by invoking the relevant NvSciC2cPcie programming interfaces.
- > Unloading of NvSciC2cPcie Linux kernel modules is not supported.
- > Error-handling of NvSciC2cPcie transfers other than PCIe AER and PCIe EDMA transfer error, leads to timeouts in the software layers exercising NvSciC2cPcie.
- > For NVIDIA Drive Recorder, C2C verification occurred by following boot order sequence - Orin-A followed by Orin-B
- > Chip to Chip communication accepts a maximum of 1022 NvSciBufObjects and 1022 NvSciSyncObjects for NvStreams over Chip to Chip communication permitting system limits.

6.7 NvMedia

This topic provides instructions for connecting and integrating the cameras as well as explaining the requirements for applications to set additional flags when creating images and videos.

Connecting and Integrating the Cameras

Multiple video and camera ports are provided in the NVIDIA DRIVE® OS Platform. Before using these cameras with the NvMedia sample applications, you must attach the cameras to the ports in a specific order. If you fail to do so, NvMedia reports errors.



Warning:

Before connecting/disconnecting cameras to/from the NVIDIA DRIVE OS platform, the system power must be disconnected. Failure to do so may damage the camera.

6.7.1 Connecting Cameras

Detailed instructions are provided at:

- > [Setting Up the Cameras](#)

6.7.2 Mapping GMSL Cameras to the SoC

Detailed instructions are provided at:

- > [Mapping Connectors to Tegra](#)

6.7.3 Understanding NvMedia

NvMedia provides APIs that process images and video. It also provides APIs for visual programming and image pipelines.

6.7.3.1 Sequence of Tasks

The following diagram illustrates the sequence of tasks that applications follow when using NvMedia DLA.



The following sections describe the sequence of tasks in the flow diagram above.

6.7.3.1.1 Setup

Allocate, configure, and set up all resources before attempting any runtime calls. Due to safety requirements, NVIDIA does not recommend calling these setup APIs during runtime.

1. Create an instance.

Configure and create an instance of a specified DLA hardware engine using `NvMediaDlaCreate()`.



Note:

A maximum of sixteen (16) NvMedia DLA instances can be created.

2. Initialize the instance.

Initialize the instance with the instance ID and number of tasks with API `NvMediaDlaInit()`.

3. Create the loadable.

Create a loadable opaque handle with `NvMediaDlaLoadableCreate()`. The handle is populated when the loadable is loaded into the instance.

4. Load the loadable.

Load a binary loadable into a DLA instance with the provided APIs in the following order:

```
NvMediaDlaAppendLoadable  
NvMediaDlaLoadLoadable
```

Only one loadable can be appended to the instance. Clients must call `NvMediaSetCurrentLoadable` to specify which loadable to work on. TensorRT builder creates the binary loadable.



Note: DLA has access to all of the available DRAM system memory. Please refer to TensorRT documentation [here](#) for further details on configuring the size of the memory pools allocated to each DLA loadable.



Note: Upon successfully compiling loadables from the given network, the TensorRT Builder reports the number of subnetwork candidates that were successfully compiled into loadables, as well as the total amount of memory used per pool by those loadables. Please refer to TensorRT documentation [here](#) for further details

5. Register buffers.

Register all buffers that will be used with the instance. Registration API is `NvMediaDlaDataRegister`.

6. Fill the `NvSciSync` attribute list.

The DLA instance fills in the `NvSciSync` attributes to provided memory with `NvMediaDlaFillNvSciSyncAttrList`.

7. Register `NvSciSync`.

Register all `NvSciSync` objects that will be used with the instance. Registration API is `NvMediaDlaRegisterNvSciSyncObj`.

8. Set the `NvSciSync` object.

Set end-of-frame (EOF) NvSciSync object or start-of-frame (SOF) NvSciSync object to the instance, if needed. The related APIs are `NvMediaSetNvSciSyncObjForSOF` and `NvMediaSetNvSciSyncObjForEOF`.

6.7.3.1.2 Runtime

Run the inference with input data on the provided loadable.

1. Insert the previous NvSciFence.

Insert the previous NvSciFence to the instance. The operation is blocked until the expiration of the previous NvSciFence. The related API is `NvMediaDlaInsertPreNvSciSyncFence`.

2. Submit.

Submit a task with specified inputs and outputs to the hardware engine with `NvMediaDlaSubmit`. To submit a task and skip execution on the DLA hardware, use `NvMediaDlaSubmitBypass`.

These are non-blocking calls. Applications can choose to block and wait for an operation on a particular buffer to complete. To do so, use the following `NvMediaTensorGetStatus` function, as applicable.

3. Get NvSciFence.

Get the end-of-frame (EOF) or start-of-frame (SOF) from the instance. The expiration of EOF indicates the completion of the operation and the expiration of SOF indicates the start of the operation. The related APIs are `NvMediaDlaGetEOFNvSciSyncFence` and `NvMediaDlaGetSOFNvSciSyncFence`.

6.7.3.1.3 Destroy

Free all resources.

1. Unregister NvSciSync objects.

Unregister all registered NvSciSync objects using `NvMediaDlaUnregisterNvSciSyncObj`.

2. Unregister data.

Unregister all registered buffers using `NvMediaDlaDataUnregister`.

3. Remove the loadable.

Remove the loaded loadable from the instance with `NvMediaDlaRemoveLoadable`.

4. Destroy the loadable handle.

Destroy the created loadable handle with `NvMediaDlaLoadableDestroy`.

5. Destroy the instance.

Destroy the hardware engine instance using `NvMediaDlaDestroy` to free all resources.

6.7.3.2 Supported Tensor Formats

The following table describes DLA support for NvMediaTensor format types.

Input Tensor Formats	Output Tensor Formats	Precision Mode
NHWC	NCHW or NC/xHWx	INT8
NCHW	NCHW or NC/xHWx	INT8
NC/xHWx	NCHW or NC/xHWx	INT8
NHWC	NCHW or NC/xHWx	FP16
NCHW	NCHW or NC/xHWx	FP16
NC/xHWx	NCHW or NC/xHWx	FP16

6.7.3.3 Image Processing and Management

The NvMedia domain performs the following processes:

- Handles Image Surface data (H/W buffer) in the form of NvSciBufObj. For example, YUV, RGB, RAW (progressive only).
- Supports registering NvSciBufObj for processing.
- Supports per-image specific metadata as part of the NvSciBufObj.

6.7.3.3.1 Image 2D

The NvMedia Image 2D component supports image surface processing features such as image copy, image scaling, image cropping. It operates on YUV/RGB input and output surfaces. It also performs format conversion to/from YUV to RGB and supports aggregated image handling.

6.7.3.3.2 Image Encode Processing (IEP)

The NvMedia Image Encode (IEP) component supports encoding the incoming NvMedia Image (YUV) inputs to H.264, H.265, and AV1.

NvMedia Image Encode Processing (IEP) APIs for NVENC supports the following features:

For H.264 encoding, the encoder has the following features:

- Accepts YUV frames as input
- Encoding common resolutions up to 3840 x 2160
- Supports H.264 Baseline, Main, and High profiles with level up to 5.2
- Provides frame-only encoding

- For I-frame or I/P-frame encoding, the driver handles the picture type according to the Group of Pictures (GOP) and IDR period.
- For I/B/P encoding, the picture reordering is handled in application code by assigning the picture type, and then sending it to the driver for encoding.
- > Supports all intra-macroblock types (16 x 16, 8 x 8, 4 x 4, PCM) and prediction types.
- > Supports inter-macroblock partition sizes from 16 x 16, 16 x 8, 8 x 16 down to 8 x 8, and skip and direct B-modes.
- > Supports disable, temporal, or spatial direct mode for:
 - B-picture
 - One reference picture for P-picture
 - Two reference pictures for B-picture
- > Supports multiple rate-control modes including:
 - Constant QP
 - Constant Bit Rate (CBR) single-pass
 - Variable Bit Rate (VBR)
 - VBR with minimum QP
- > Supports dynamic slice mode based on byte size and static multiple slices in the frame.
- > Supports intra-refresh mode with refresh period, and instant refresh P-picture.
- > Supports adaptive 8x8 transform mode.
- > Supports VUI and SEI insertion.
- > Supports CAVLC and CABAC.
- > Supports rotation/mirroring mode.
- > Supports dynamic configure changes at the frame level:
 - SPS PPS output on next frame
 - Constrained frame encode
 - Instant intra refresh P picture
 - New SEI packet insertion
 - GOP length and IDR period update
 - Rate control mode change

For H.265 encoding, the encoder has the following features:

- > Accepts YUV frames as input.
- > Encoding common resolutions up to 3840 x 2160
- > Supports H.265 Main profiles with level up to 6.0.
- > Supports Frame only encoding.
- > For I only or I/P encoding, the driver handles the picture type according to the GOP and IDR period. B pictures are not supported.
- > Supports all intra CU types (32 x 32, 16 x 16, 8 x 8, 4 x 4, PCM) and prediction types.
- > Inter CU partition sizes from 32x32 to 8x8, with partition mode of PART_2Nx2N, PART_2NxN, PART_Nx2N, PART_2NxN, PART_nLx2N, PART_nRx2N plus skip.

- > 1 reference picture for P picture.
- > Multiple rate-control modes—constant QP, CBR (single-pass), VBR, VBR with min QP.
- > Dynamic slice mode based on byte size and static multiple slices in the frame.
- > Intra refresh mode with refresh period, and instant refresh P picture.
- > VUI and SEI insertion.
- > CABAC only.
- > Dynamic configure change at frame level:
 - SPS PPS output on next frame
 - Constrained frame encode
 - Instant intra refresh P picture
 - New SEI packet insertion
 - Gop length and IDR period update
 - Rate control mode change

For AV1 encoding, the encoder has the following features:

- > Accepts YUV frames as input (YUV420 only).
- > Encoding common resolutions up to 7680 x 4320
- > Supports main profile up to level 6.3
- > Supports 8/10 bit encoding
- > Supports Frame only encoding.



Note: Processing of a submitted task can potentially time out for tasks that do not complete in real time within an internally defined timeout (`mLocktimeout`). This implies that the applied configuration (such as resolution and bitrate) is not supported on the platform at the configured hardware clocks. This behavior shows up as failed/errored-out tasks even when the supplied configuration is valid.

6.7.3.3.3 Image JPEG Encode (IJPE)

- > Baseline ITU-T T.81 / ISO/IEC 10918-1
- > Programmable Huffman Table
- > Programmable Quant Table
- > 8bit YUV420 semi-planar Input Format
- > 8bit YUV420 planar input format
- > Rate control i.e. target image size
- > Quality factor parameter from 1 (lowest quality) to 100 (highest quality)



Note: Processing of a submitted task can potentially time out for tasks that do not complete in real time within an internally defined timeout (`mLocktimeout`). This implies that the applied configuration (such as resolution and bitrate) is not supported on the platform at the configured hardware clocks. This behavior shows up as failed/errored-out tasks even when the supplied configuration is valid.

6.7.3.3.4 Image JPEG Decode (IJPD)

- > Baseline ITU-T T.81 / ISO/IEC 10918-1
- > As the output format depends on the format of the input stream, the following output formats are possible (depending on the input stream):

Input bit-stream	Output color format	Output memory layout format	Output pixel packing format	Output bit depth	Comment
YUV420	YUV420	pitch linear	planar	8-bit	
	YUV420	pitch linear	semi-planar	8-bit	
YUV422	YUV422H/V	pitch linear	planar	8-bit	
		pitch linear	YUY2	8-bit	
YUV444	YUV444	pitch linear	planar	8-bit	
Monochrome	YUV400	pitch linear	planar	8-bit	
RGBA	RGBA	pitch linear	interleaved	8-bit	A can be either 0x00 or 0xFF
	BGRA	pitch linear	interleaved	8-bit	A can be either 0x00 or 0xFF
	ABGR	pitch linear	interleaved	8-bit	A can be either 0x00 or 0xFF
	ARGB	pitch linear	interleaved	8-bit	A can be either 0x00 or 0xFF

- > Support for all input formats to give output in RGBA where R, G, B, A locations are programmable (supported value for A can be either 0x00 or 0xFF).
- > The supported chroma subsample source and target formats are as follows:

Source format	Target format after chroma sub-sample conversion
YUV420	YUV422H
YUV422V	YUV422H
	YUV420

YUV422H	YUV420
YUV444	YUV422H
	YUV420

- Support down-scaling of decoded output by a power of 2. The output size downscale by 2/4/8 factor in both width and height. For example, input with resolution (w x h) can output in any one of these output resolutions (w/2 x h/2) or (w/4 x h/4) or (w/8 x h/8).
- Support for decoding of YUV444 non-interleaved JPEG stream. Output produced will be in YUV444, 8-bit, pitch linear planar format.



Note: Processing of a submitted task can potentially time out for tasks that do not complete in real time within an internally defined timeout (`mLocktimeout`). This implies that the applied configuration (such as resolution and bitrate) is not supported on the platform at the configured hardware clocks. This behavior shows up as failed/errored-out tasks even when the supplied configuration is valid.

6.7.3.3.5 Image LDC

The NvMedia Image LDC component supports image surface processing features such as lens distortion correction and temporal noise reduction.

6.7.3.3.6 Optical Flow Accelerator (OFA)

NVIDIA Optical Flow Accelerator (OFA) is hardware accelerator for computing optical flow and stereo disparity between the frames.

Optical flow is useful in various use-case such as object detection and tracking, while Stereo disparity is used in depth estimation.

The hardware capabilities of OFA are exposed through NvMedia IOFA APIs.

OFA can operate in two modes:

- Stereo Disparity Mode: In this mode, OFA generates disparity between rectified left and right images of stereo capture.
- Optical Flow Mode: In this mode, OFA generates optical flow between two given frames, returning X and Y component of flow vectors.

OFA generates disparity / flow vector block-wise, one output for each input block of 8x8 / 4x4 / 2x2 / 1x1 pixels (referred as output grid size). The generated output can be further post-processed to improve accuracy, up sampled to produce dense map.

Stereo Disparity Mode

OFA processes rectified left and right view of stereo captures and generates disparity values between them.

OFA in Stereo Disparity Mode, can process only rectified stereo image pairs. Rectification of stereo image pair can be done using NvMedia Image 2D.

The output stereo disparity format is fixed signed 10.5 (2 bytes per disparity output) and range of disparity is fixed [0, 127] or [0, 255]. We need to divide the output values by 32 to get a disparity value in terms of pixel units.

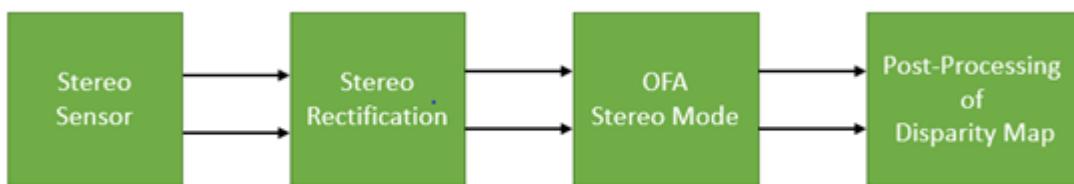
OFA also provides Cost buffer which contain HW cost of winner disparity candidates. Cost for each output is 8 bit in size and range is 0 – 255.

Disparity values have inverse relationship with depth of the objects (distance between object and sensor).

Stereo Parameter	Description
Input Image Size	Minimum Size – 32 x 32 Maximum Size – 8192 x 8192 No alignment requirement
Input Image format / bit depth	Luma / Single channel Input Supports bit depth of 8/10/12/16 bits
Disparity Output	Disparity Map in fixed S 10.5 format
Cost Output	HW cost for winner disparity candidate
Output Grid Size	1x1/2x2/4x4/8x8
Maximum Disparity Range	128 / 256
Search Direction	Left / Right Disparity Map
Region Of Interest Support	Supports maximum 32 ROI per stereo pair

Stereo Use Case

The high-level block diagram of OFA Stereo use case is as follows



The stereo sensor provides unrectified left and right view images.

The application uses NvMedia 2D API to rectify these input stereo image pair.

OFA, configured in stereo mode processes rectified stereo image pair and generates disparity map and HW cost buffer.

The post-processing module processes the generated disparity map and HW cost buffer to improve accuracy of the disparity map.

Common examples of post processing algorithms are:

- > Median filtering of disparity map
- > Up sampling to produce a dense disparity map
- > left-right consistency check

Stereo rectification and post-processing operations are not supported using OFA and must be implemented on other HW accelerators.

Optical Flow Mode

OFA generates optical flow between two given frames.

The input to OFA in this mode, is image pyramid of input and reference frames. As search range of single layer is small, pyramid approach is required to track large motion. Each pyramid level will search around output of previous pyramid level.

OFA can process image pyramid generated with fixed scale factor of 2. Image pyramid generation needs be done outside OFA.

OFA generates flow vector for input block specified by output Grid size. Each flow vector has X and Y component which represent motion in X and Y direction.

The output flow format is fixed signed 10.5 (4 bytes per flow vector). We need to divide the output values by 32 to get a disparity value in terms of pixel units.

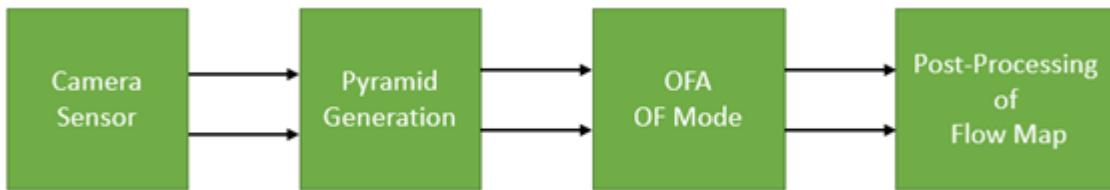
OFA also provides Cost buffer which contain HW cost of winner flow candidate for each output. Cost for each output is 8 bit in size and range is 0 – 255.

OF Parameter	Description
Input Image Size	Minimum Size – 32 x 32 Maximum Size – 8192 x 8192 No alignment requirement
Input Image format / bit depth	Luma / Single channel Input Supports bit depth of 8/10/12/16 bits
Max Pyramid Levels	5
Flow Output	Flow Map (mvx, mvy) in fixed S10.5 format

OF Parameter	Description
Cost Output	HW cost for winner disparity candidate
Output Grid Size	1x1/2x2/4x4/8x8
Region Of Interest Support	Supports maximum 32 ROI per stereo pair

Optical Flow Use Case

The high-level block diagram of OFA optical flow use case is as follows



Here stereo sensor provides the input and reference frame.

The application generates the image pyramid of input and reference frames.

OFA, configured in off mode processes pyramids of input and reference frames and generates the flow map and HW cost buffer.

The post-processing module processes the generated flow map and HW cost buffer to improve accuracy of the generated flow map.

Common examples of post-processing algorithms are:

- Median filtering of flow map
- Up sampling to produce dense flow map
- Forward-backward consistency check

Pyramid generation and post-processing operations are not supported using OFA and must be implemented on other HW accelerators.



Note: Processing of a submitted task can potentially time out for tasks that do not complete in real time within an internally defined timeout (`mLocktimeout`). This implies that the applied configuration (such as resolution and bitrate) is not supported on the platform at the configured hardware clocks. This behavior shows up as failed/errorred-out tasks even when the supplied configuration is valid.

6.7.3.3.7 SIPL

The NvMedia SIPL framework provides high dynamic range (HDR) camera processing which outputs images for human and machine vision. It handles individual camera processing or multiple cameras connected to an image aggregator chip.

6.7.3.4 NvMedia Tensor

This topic explains how to use the NvMedia Tensor API.

NvMedia Tensors are multi-dimensional data structures that NvMedia creates in SoC DRAM memory to store multi-dimensional arrays of a specific data type. For example: integers, float etc.

This topic assumes a basic understanding of NvSciBuf APIs. See [Buffer Allocation](#) for more information.

6.7.3.4.1 Types of Tensors

Currently, NvMedia only supports 4-dimensional tensors:

NvMedia Tensors are used with NvMedia DLA components.

NvMedia Tensors can be created by allocating NvSciBuf through NvMedia Tensor attributes using [NvSciBuf API](#). As NvSciBuf APIs facilitate data sharing between NvMedia and NVIDIA® CUDA®, this allows tensors allocated to be reused as permitted by NvSciBuf API. For more information, see the NvSciBuf API and use cases.

NvMedia Tensor have two types of attributes:

- > **Tensor format attributes** describe a tensor's order and format in memory.
- > **Tensor allocation attributes** describe additional properties of a tensor, such as:
 - Width, height, channels, and number of tensor surfaces.
 - CPU access mapping (cached/uncached/unmapped).
 - Shared memory space across virtual machine partitions.

6.7.3.4.1.1 Tensor Format Attributes

The following sections describe the tensor format attributes that NvMedia Tensors may have.

NVM_TENSOR_ATTR_DATA_TYPE Attribute

This attribute specifies the tensor type. The value may be:

`NVM_TENSOR_ATTR_DATA_TYPE_UINT`

Indicates tensor of unsigned integers data types.

`NVM_TENSOR_ATTR_DATA_TYPE_INT`

Indicates tensor of signed integers data types.

`NVM_TENSOR_ATTR_DATA_TYPE_FLOAT`

Indicates tensor of float data types.

NVM_TENSOR_ATTR_DIMENSION_ORDER Attribute

Specifies the layout and order of the tensor elements.

4D Tensor layout includes N, C, H, and W dimensions, where N refers to the number of surfaces (or batch size), C refers to the number of channels in the surface (for example, RGB if the surface type is an image), H refers to the height of the surface, and W refers to the width of the surface.

The following are possible values for 4D tensor formats:

- > `NVM_TENSOR_ATTR_DIMENSION_ORDER_NCHW`
 - > `NVM_TENSOR_ATTR_DIMENSION_ORDER_NHWC`
 - > `NVM_TENSOR_ATTR_DIMENSION_ORDER_NCxHWx`
- `NVM_TENSOR_ATTR_BITS_PER_ELEMENT` Attribute

Specifies the number of bits per element. The value can be:

- > `NVM_TENSOR_ATTR_BITS_PER_ELEMENT_64`
Indicates that each element is 64 bits wide.
- > `NVM_TENSOR_ATTR_BITS_PER_ELEMENT_32`
Indicates that each element is 32 bits wide.
- > `NVM_TENSOR_ATTR_BITS_PER_ELEMENT_16`
Indicates that each element is 16 bits wide.
- > `NVM_TENSOR_ATTR_BITS_PER_ELEMENT_8`
Indicates that each element is 8 bits wide.

6.7.3.4.1.2 Tensor Allocation Attributes

The following sections describe the tensor allocation attributes that NvMedia Tensor may have.

`NVM_TENSOR_ATTR_ALLOC_TYPE` Attribute

Specifies the allocation type for the tensor. The value may be:

- > `NVM_TENSOR_ATTR_ALLOC_NONE`
Indicates tensor allocation on SoC DRAM.

`NVM_TENSOR_ATTR_4D_N`

Specifies the number of tensor surfaces in a tensor. It is required to determine the size of memory to be allocated.

`NVM_TENSOR_ATTR_4D_C`

Specifies the number of tensor channels. It is required to determine the size of memory to be allocated.

`NVM_TENSOR_ATTR_4D_H` and `NVM_TENSOR_ATTR_4D_W` Attributes

Specifies the width and height of the tensor. It is required to determine the size of memory to be allocated.

`NVM_TENSOR_ATTR_4D_X`

Specifies the interleaving factor of tensor surfaces (only in NCxHWx tensor ordering). It is required to determine the size of memory to be allocated.

NVM_TENSOR_ATTR_CPU_ACCESS Attribute

Specifies the coherency policy to use for accesses of the tensor from the CPU. The value may be:

- > NVM_TENSOR_ATTR_CPU_ACCESS_UNCACHED

Specifies that accesses from CPU never cache data.

Setting this attribute results in the following behavior: While writing to the tensor buffers from the CPU using `NvMediaTensorLock()` and `NvMediaTensorUnlock()`, NvMedia uses appropriate memory barriers before handing over the tensor buffer to hardware engines to ensure coherency.

- > NVM_TENSOR_ATTR_CPU_ACCESS_CACHED

Specifies that accesses from the CPU can pass through caches and store buffers.

Setting this attribute results in the following behavior:

- While reading the tensor from the CPU using `NvMediaTensorLock()` and `NvMediaTensorUnlock()`, caches are invalidated as necessary to ensure that the CPU gets the latest data written by the hardware engines.
- While writing the tensor from the CPU using `NvMediaTensorLock()` and `NvMediaTensorUnlock()`, caches are flushed as necessary before handing over the tensor buffers to hardware engines to ensure coherency.

In both cases, the tensor memory is mapped and it can be accessed with a mapping into the current process's virtual address space.

- > NVM_TENSOR_ATTR_CPU_ACCESS_UNMAPPED

Specifies a coherency policy that is the same as for `NVM_TENSOR_ATTR_CPU_ACCESS_UNCACHED`. However, the tensor is not mapped into the current process's virtual address space.

If the attribute is not specified, the coherency policy defaults to `NVM_TENSOR_ATTR_CPU_ACCESS_UNCACHED`.

6.7.3.4.2 Tensor API Functions

This section describes NvMedia Tensor API functions that create handles from NvSciBuf, destroy, and manage tensors.

6.7.3.4.2.1 NvMedia Tensor Creation and Destroy Functions

These API functions allow the creation and destruction of tensors.

`NvMediaTensorCreateFromNvSciBuf()`

Creates an NvMedia Tensor handle from an NvSciBuf created with the NvSciBuf API, after the required NvSciBuf attributes list is prepared.

Every hardware engine in an NVIDIA SoC can have a different alignment or stride constraints. Hence, sharing a buffer across various engines requires that buffer allocation satisfies the constraints of all of the engines that share the buffer. An engine whose constraints are not satisfied may fail to operate on the buffer. The allocation functions provided by the various NvMedia drivers only satisfy the constraints of the engines that are visible to them, and so cannot be used to allocate shared buffers.

NvSciBuf is a buffer allocation module that satisfies a common set of constraints that are compatible with all of the hardware engines. It thus can allocate buffers that are shareable across the hardware engines visible to various drivers.

This is a typical flow to allocate an NvSciBufObj, which can be mapped to an NvMediaTensor:

1. The application creates an NvSciBufAttrList.
2. The application queries NvMedia to fill the NvSciBufAttrList by passing a set of NvMediaTensor allocation attributes and an NvMediaType as input to NvMediaTensorFillNvSciBufAttrs().
3. The application may set any of the public NvSciBufAttribute values that NvMedia does not set.

For more details on NvSciBuf concepts, terminology, and the API, see [Buffer Allocation](#).

The following NvSciBuf input attributes are set by NvMedia, and must not be set by the application:

- > NvSciBufGeneralAttrKey_Types
- > NvSciBufGeneralAttrKey_NeedCpuAccess
- > NvSciBufGeneralAttrKey_EnableCpuCache
- > NvSciBufTensorAttrKey_DataType
- > NvSciBufTensorAttrKey_NumDims
- > NvSciBufTensorAttrKey_SizePerDim
- > NvSciBufTensorAttrKey_AlignmentPerDim
- > NvSciBufTensorAttrKey_StridesPerDim
- > NvSciBufTensorAttrKey_PixelFormat
- > NvSciBufTensorAttrKey_BaseAddrAlign

The following attributes are not set by NvMedia and must be set by the application:

- > NvSciBufGeneralAttrKey_RequiredPerm
4. If the same NvSciBufObj object has to be shared with other user mode drivers (UMDs), the application can get the corresponding NvSciBufAttrList from the respective UMDs.
 5. The application asks NvSciBuf to reconcile all of the filled NvSciBufAttrList objects, then allocates an NvSciBuf object.
 6. The application queries NvMedia to create an NvMediaTensor from the allocated NvSciBuf object by calling NvMediaTensorCreateFromNvSciBuf().

- The NvMediaTensor can be passed as input and output to any of the NvMedia API functions that accept an NvMediaTensor as a parameter.

Example: NvMedia Tensor Allocation with NvSciBuf

Following is an example of how to allocate an NvMedia Tensor with NvSciBuf:

```

NvMediaStatus status;
NvSciError err;
NvSciBufModule module;
NvSciBufAttrList attrlist;
NvSciBufAttrList conflictlist;
NvSciBufObj bufObj;
NvMediaTensor *tensor;
uint32_t numTensorAttr = NVM_TENSOR_ATTR_MAX;
NVM_TENSOR_DEFINE_ATTR(tensorAttr);
/*NvMedia related initialization. */

status = NvMediaTensorNvSciBufInit();
/*NvSciBuf related initialization. */
err = NvSciBufModuleOpen(&module);
NvSciBufAttrKeyValuePair attr_kvp = {NvSciBufGeneralAttrKey_RequiredPerm, &access_perm,
                                     sizeof(access_perm)};
/*Create NvSciBuf attribute list. */
err = NvSciBufAttrListCreate(module, &attrlist);
err = NvSciBufAttrListSetAttrs(attrlist, &attr_kvp, 1);
/* Initialize tensorAttrs as required. */
NVM_TENSOR_SET_ATTR_4D(tensorAttr, n, c, h, w, NCHW, INT, 8, UNCACHED, NONE, x);
/* Ask NvMedia to fill NvSciBufAttrs corresponding to
tensorAttrs. */
status = NvMediaTensorFillNvSciBufAttrs(NULL,
                                         tensorAttr,
                                         numTensorAttr,
                                         0,
                                         attrlist);
/* Reconcile the NvSciBufAttrs and then allocate an NvSciBufObj. */
err = NvSciBufAttrListReconcileAndObjAlloc(&attrlist, 1, bufobj, &conflictlist);
/* Create NvMediaTensor from NvSciBufObj. */
status = NvMediaTensorCreateFromNvSciBuf(NULL, bufobj, &tensor);
/* Free the NvSciBufAttrList which is no longer required. */
err = NvSciBufAttrListFree(attrlist);
/* Use the tensor as input or output as supported. */
.....
.....
/* Free the resources after use. */
/* Destroy NvMediaTensor. */
NvMediaTensorDestroy(tensor);
/* NvMedia related Deinit. */
NvMediaTensorNvSciBufDeinit();

/* NvSciBuf related_deinit. */
NvSciBufObjFree(bufobj);
NvSciBufModuleClose(module);

```

Example: Reconcile between NvMediaTensor and Image Attributes (Optional)

This example shows how to reconcile NvMediaTensor and Image attributes.

1. Create Image attributes unreconciled_attrlistImage. For more information, see [Multi Datatype Attribute Lists Reconciliation](#)
2. Reconcile Image and NvMediaTensor attributes.

```
attr[0] = unreconciled_attrlistImage;
attr[1] = unreconciled_attrlistTensor;
err = NvSciBufAttrListReconcileAndObjAlloc(&attrlist,
2, bufobj, &conflictlist);
NvMediaTensorDestroy()
```

Destroys a previously allocated NvMedia Tensor object.

Example:

```
if (tensor) {
    NvMediaTensorDestroy(tensor);
}
```

6.7.3.5 NvSciSync

The NvMedia APIs provide true hardware acceleration of image processing using hardware engines on NVIDIA DRIVE AGX devices.

NvSciSync extends the NvMedia imaging components to support synchronization among the imaging components, NVIDIA® CUDA® components, and NvSciSync-based applications. NvSciSync supports imaging components that are targeted for Advanced Driver Assistance Systems (ADAS) and autonomous application development. They do not support NvMedia video components.

Imaging components that have NvSciSync extensions support:

- Accepting an NvSciSyncFence object when the NvMedia engine is used as a waiter
- Returning an NvSciSyncFence object when the NvMedia engine is used as a signaler

An NvSciSyncFence object is a snapshot of an NvSciSync object's state.

6.7.3.5.1 For Additional Information

See these sources for additional information about NvSciSync and related topics:

- NvSciSync: [NvStreams Synchronization](#)
- NvMedia image processing and the components of the NvMedia image processing pipeline: the [NvMedia Architecture](#) topic in this document

6.7.3.5.2 Definitions

This section defines some terms that are specific to NvSciSync operations to NvMedia.

- **Frame:** The smallest unit of data that a hardware engine acceleration API can operate on.

- > **SOF Fence:** Start of frame fence, a fence whose expiry indicates the start of engine processing.
- > **EOF Fence:** End of frame fence, a fence whose expiry indicates that processing is complete and an output frame is ready to be used.
- > **PRE Fence:** A fence on which the start of engine processing is blocked until the expiry of the fence.

6.7.3.5.3 NvSciSync Functions for Specific Imaging Components

This topic provides an overview of NvSciSync functions supported for NvMedia imaging components (2D, ICP, and so on).

NvSciSync supports a group of functions for several imaging components, such as NvMedia2D and NvMediaICP. The groups of functions have parallel names. For example, **NvMedia XX RegisterNvSciSyncObj()** registers an NvSciSync object with a component, where XX is 2D for NvMedia2D, ICP for NvMediaICP, etc.

NvSciSync supports the following functions for each component, with a few exceptions detailed below:

- > Query a component's NvSciSyncObj attributes for waiting or signaling.

Call **NvMediaXXFillNvSciSyncAttrList()** to query the NvSciSync attributes of component XX. The function sets (fills) the attributes in memory provided by the caller. NvSciSync objects allocated with such NvSciSyncAttrList objects are only accepted by component XX functions.

- > Register and unregister an NvSciSync object.

Call **NvMedia XX RegisterNvSciSyncObj()** to register NvSciSyncObj objects with component XX. You must register every NvSciSyncObj that component XX is to use.

During teardown, call **NvMediaXXUnRegisterNvSciSyncObj()** to unregister the registered NvSciSyncObj objects with component XX.

- > Set an NvSciSyncObj object for SOF usage with the component.

Call **NvMedia XX SetNvSciSyncObjForSOF()** to tell component XX to use a particular NvSciSyncObj for signaling start of frame (SOF). You must call this function before you call any of the component's main image processing functions.

- > Set an NvSciSyncObj for EOF usage with the component.

Call **NvMedia XX SetNvSciSyncObjForEOF()** to tell component XX to use a particular NvSciSyncObj for signaling end of frame (EOF). You must call this function before you call any of the component's main image processing functions.

- > Wait for an NvSciSyncFence.

Call **NvMediaXXInsertPreNvSciSyncFence()** to tell component XX to wait on an NvSciSyncFence before actually starting image processing. You must call this function before you call any of the component's main image processing functions.

- > Get an NvSciSyncFence SOF.

Call `NvMediaXXGetSOFNvSciSyncFence()` to get an `NvSciSyncFence` from component XX whose expiry indicates that the last submitted image processing request has started. You must call this function only after you call the component's main image processing functions.

- Get an `NvSciSyncFence` EOF.

Call `NvMediaXXGetEOFNvSciSyncFence()` to get an `NvSciSyncFence` from component XX whose expiry indicates that the last submitted image processing request has completed. You must call this function only after you call the component's main image processing functions.

The functions are implemented for each of the following NvMedia imaging components.

Component	Notes
2D	
LDC	
SIPL	the ISP hardware engine does not generate any fence to indicate the start of a frame.
IEP	
OFA	
DLA	

Not every function is implemented for every imaging component. The following table shows functions implemented for each supported component. For SIPL APIs, refer to the *NVIDIA DRIVE OS API Reference Guide*.

Generic function	2D	LDC	IEP	OFA	DLA
Description					
<code>NvMediaXXFillNvSciSyncAttrList()</code> Fills attributes in an <code>NvSciSyncAttrList</code> .	X	X	X	X	X
<code>NvMediaXXRegisterNvSciSyncObj()</code> Registers an <code>NvSciSyncObj</code> with a component object.	X	X	X	X	X

Generic function	2D	LDC	IEP	OFA	DLA
Description					
NvMediaXXUnregisterNvSciSyncObj() Registers an NvSciSyncObj with a component object.	X	X	X	X	X
NvMediaXXSetNvSciSyncObjforSOF() Specifies an NvSciSyncObj to use as an SOF NvSciSyncFence.	—	—	—	—	X
NvMediaXXSetNvSciSyncObjforEOF() Specifies an NvSciSyncObj to use as an EOF NvSciSyncFence.	X	X	X	X	X
NvMediaXXInsertPreNvSciSyncFence() Sets an NvSciSyncFence as a preference.	X	X	X	X	X
NvMediaXXGetSOFNvSciSyncFence() Gets an SOF NvSciSyncFence for an NvMediaXXProcess() operation.	—	—	—	—	X
NvMediaXXGetEOFNvSciSyncFence() Gets an EOF NvSciSyncFence for an NvMediaXXProcess() operation.	X	X	X	X	X

6.7.3.5.4 Code Examples

Following is example code that illustrates NvSciSync operations for the 2D component. It can also be used as a model for NvSciSync operations in the other imaging components.

```
***** Init Time *****/
NvSciSyncModule    nvscisyncModule;
NvSciError         nverr;
NvSciSyncAttrList nvscisyncattr_w;
NvSciSyncAttrList nvscisyncattr_s;
NvSciSyncAttrList nvscisyncattr_unreconciled_h[2];
NvSciSyncAttrList nvscisyncattr_reconciled;
NvSciSyncAttrList ConflictAttrList;
NvSciSyncFence    eofnvscisyncfence = NV_SCI_SYNC_FENCE_INITIALIZER;
NvSciSyncObj      nvscisyncEOF, nvscisyncpre;
NvMediaStatus     nvmstatus;
NvMedia2D *nvm2dhdl = NULL;
```

```

nvmstatus = NvMedia2DCreate(&nvm2dhd1, NULL);
nverr = NvSciSyncModuleOpen(&nvscisyncModule);
***** NvMedia 2D as a signaler *****/
nverr = NvSciSyncAttrListCreate(nvscisyncModule, &nvscisyncattr_s);
nvmstatus = NvMedia2DFillNvSciSyncAttrList(nvm2dhd1, &nvscisyncattr_s,
    NVMEDIA_SIGNALER);
nvscisyncattr_unreconciled_h[0] = nvscisyncattr_s;
nvscisyncattr_unreconciled_h[1] = get attribute list from the appropriate waiter;
nverr = NvSciSyncAttrListReconcile(nvscisyncattr_unreconciled_h[],
    2 , &nvscisyncattr_reconciled, &ConflictAttrList);
nverr = NvSciSyncObjAlloc(nvscisyncattr_reconciled, &nvscisyncEOF);
***** NvMedia 2D as a waiter *****/
nverr = NvSciSyncAttrListCreate(&nvscisyncattr_w);
nvmstatus = NvMedia2DFillNvSciSyncAttrList(nvm2dhd1, &nvscisyncattr_w, NVMEDIA_WAITER);
/*If the signaler is also in the same process as the 2D Waiter, then
NvSciSyncAttrListReconcileAndObjAlloc or NvSciSyncAttrListReconcile and
NvSciSyncObjAlloc API pair has/have to be used to allocate nvscisyncpre NvSciSyncObject.
If the signaler is in a different process/VM than the 2D Waiter, then
NvSciSyncAttrList export/import APIs and NvSciSyncObjIpc Export/Import APIs
have to be used allocate a NvSciSyncObject on signaler and waiter sides.
nvscisyncpre is the imported NvSciSyncObject on the waiter side */
/*All the NvSciSyncObjects(NvSciSyncObjects associated with PreFences, EOFFence
) which will be used by NvMedia2D must be registered upfront. */
***** Start of registration of NvSciSync objects *****/
nvmstatus = NvMedia2DRegisterNvSciSyncObj(nvm2dhd1, NVMEDIA_EOFSYNCOBJ, nvscisyncEOF);
/* Register all the NvSciSync objects which will be used to generate preferences for
NvMedia2DBlit operation. nvscisyncpre is one such Pre NvSciSync object */
nvmstatus = NvMedia2DRegisterNvSciSyncObj(nvm2dhd1, NVMEDIA_PRESYNCOBJ, nvscisyncpre);
***** End of Registration of NvSciSync Objects ****.
/*Allocate a NvSciBufObj for inputs, say inputimg[N] */
/*Allocate a NvSciBufObj for output, say outputimg */
/*Register the inputimgs and outputimg */
nvmstatus = NvMedia2DRegisterNvSciBufObj(nvm2dhd1, inputimg[i]);
nvmstatus = NvMedia2DRegisterNvSciBufObj(nvm2dhd1, outputimg);
***** End of init-time and start of run-time ****/
/* Acquire an empty NvMedia 2D parameters object */
NvMedia2DComposeParameters params;
nvmstatus = NvMedia2DGetComposeParameters(nvm2dhd1, &params);
nvmstatus = NvMedia2DSetNvSciSyncObjforEOF(nvm2dhd1, nvscisyncEOF);
/*Get a nvscisyncfence from somewhere(maybe a eofnvscisyncfence of
some other engine operation) which neeeds to be inserted as prefence
for 2DBlit operation. prenvscisyncfence is one such NvSciSyncFence. */
nvmstatus = NvMedia2DInsertPreNvSciSyncFence(nvm2dhd1, prenvscisyncfence);
/* Set the source layer parameters */
nvmstatus = NvMedia2DSetSrcNvSciBufObj(nvm2dhd1, params, i, inputimg[i]);
nvmstatus =
    NvMedia2DSetSrcGeometry(nvm2dhd1, params, i, NULL, NULL, NVMEDIA_2D_TRANSFORM_NONE);
nvmstatus = NvMedia2DSetSrcFilter(nvm2dhd1, params, i, NVMEDIA_2D_FILTER_OFF);
nvmstatus = NvMedia2DSetSrcBlendMode(nvm2dhd1, params, i,
    NVMEDIA_2D_BLEND_MODE_CONSTANT_ALPHA, 0.5);
/* Set destination surface */
nvmstatus = NvMedia2DSetDstNvSciBufObj(nvm2dhd1, params, outputimg);
/* Submit the compose operation */
NvMedia2DComposeResult composeResult;

```

```

nvmstatus = NvMedia2DCompose(nvm2dhd1, params, &composeResult);
nvmstatus = NvMedia2DGetEOFNvSciSyncFence(nvm2dhd1, &composeResult, &eofnvscisyncfence);
/*eofnvscisyncfence may be used as prefence for some other engine operation
or application can decide to wait on CPU till their expiry using NvSciSyncWait API. */
***** End of Run time *****/
/*Unregister the inputimgs and outputimg */
nvmstatus = NvMedia2DUnregisterNvSciBuf0bj(handle, inputimg[i]);
nvmstatus = NvMedia2DUnregisterNvSciBuf0bj(handle, outputimg);
***** Unregister registered NvSciSync objects *****/
nvmstatus = NvMedia2DUnRegisterNvSciSync0bj(nvm2dhd1, nvscisyncEOF);
nvmstatus = NvMedia2DUnRegisterNvSciSync0bj(nvm2dhd1, nvscisyncpre);
NvSciSyncAttrListFree(nvscisyncattr_w);
NvSciSyncAttrListFree(nvscisyncattr_s);
NvSciSyncAttrListFree(nvscisyncattr_reconciled);
NvSciSyncObjFree(nvscisyncEOF);
NvSciSyncObjFree(nvscisyncpre);
NvSciSyncModuleClose(nvscisyncModule);

```

6.7.4 Understanding the Sensor Input Processing Library (SIPL) Framework

The NvMedia SIPL framework provides a simplified API to capture the output of image sensors connected to NVIDIA® DRIVE AGX Orin™ platforms.

The purpose of SIPL is to abstract the following operations from the application layer:

- > Programming the image sensors, EEPROMs, serializers, and deserializers.
- > Programming platforms to capture and process the images using hardware image processing pipelines (ISPs).

6.7.4.1 Camera SIPL Notifications

ENUM	
information	NOTIF_INFO_ICP_PROCESSING_DONE
	NOTIF_INFO_ISP_PROCESSING_DONE
	NOTIF_INFO_ACP_PROCESSING_DONE
	NOTIF_INFO_CDI_PROCESSING_DONE
	NOTIF_INFO_ICP_AUTH_SUCCESS

ENUM	
Warnings	NOTIF_WARN_ICP_FRAME_DROP
	NOTIF_WARN_ICP_FRAME_DISCONTINUITY
	NOTIF_WARN_ICP_CAPTURE_TIMEOUT
Errors	NOTIF_ERROR_ICP_BAD_INPUT_STREAM
	NOTIF_ERROR_ICP_CAPTURE_FAILURE
	NOTIF_ERROR_ICP_EMB_DATA_PARSE_FAILURE
	NOTIF_ERROR_ISP_PROCESSING_FAILURE
	NOTIF_ERROR_ACP_PROCESSING_FAILURE
	NOTIF_ERROR_CDI_SET_SENSOR_CTRL_FAILURE
	NOTIF_ERROR_DESER_LINK_FAILURE
	NOTIF_ERROR_DESERIALIZER_FAILURE
	NOTIF_ERROR_SERIALIZER_FAILURE
	NOTIF_ERROR_SENSOR_FAILURE
	NOTIF_ERROR_ISP_PROCESSING_FAILURE_RECOVERABLE
	NOTIF_ERROR_ICP_AUTH_FAILURE
	NOTIF_ERROR_INTERNAL_FAILURE

6.7.4.2 SIPL Guidance on Output Image Formats

SIPL supports multiple image formats using the ICP and ISP outputs. This topic describes how the output formats are set and which parameters can be modified to override output formats under specific conditions.

ICP Output Formats

ICP output formats are dependent on the sensor used.

Based on the vendor and sensor module specifics, the output format of ICP is determined using the `PlatformCfg::DeviceBlockInfo::CameraModuleInfo::SensorInfo::VirtualChannelInfo::cfa` and `inputFormat` fields in the `PlatformCfg` struct provided as input to the `INvSIPLCamera::SetPlatformCfg` API.

Refer to `NvSIPLCapStructs.h` for the following:

1. Supported values for input format that correspond to `NvSiplCapInputFormatType` enum.
2. Supported values for `cfa` that correspond to the valid `NVSIPL_PIXEL_ORDER` flags.



Note: SIPL does not add restrictions on the use of different input format type/pixel orders because these are specific to sensors, and SIPL does not have a way of determining the expectations of each sensor. A recommendation is to adhere to the specifics of the sensors.

SIPL however does pose restrictions such as rejecting bad combinations of input format and pixel order. Some examples of the invalid combinations are as follows.

1. `NVSIPL_CAP_INPUT_FORMAT_TYPE_YUV422` and `NVSIPL_PIXEL_ORDER_RGBA`
2. `NVSIPL_CAP_INPUT_FORMAT_TYPE_RAW6` and `NVSIPL_PIXEL_ORDER_RGBA`
3. `NVSIPL_CAP_INPUT_FORMAT_TYPE_RGB888` and `NVSIPL_PIXEL_ORDER_RGGB`
4. `NVSIPL_CAP_INPUT_FORMAT_TYPE_YUV422` and `NVSIPL_PIXEL_ORDER_RGGB`
5. `NVSIPL_CAP_INPUT_FORMAT_TYPE_RAW6` and `NVSIPL_PIXEL_ORDER_YUV`
6. `NVSIPL_CAP_INPUT_FORMAT_TYPE_RGB888` and `NVSIPL_PIXEL_ORDER_YUV`

ISP Output Formats

Refer to the table provided as part of `NvSIPLCamera.hpp` for documentation on `INvSIPLCamera::RegisterImages` API for the various ISP output formats supported by SIPL.

In 6.0 release, users are expected to call `INvSIPLCamera::GetImageAttributes` API before calling `INvSIPLCamera::RegisterImages` API in order to validate the attributes provided by the user. The sequence of API calls is as follows:

1. Application calls `NvSciBufAttrListCreate` to create an unreconciled buffer attribute list.

- Application implements and calls `OverridelImageAttributes(NvSciBufAttrs)` function to set the required output format for each ISP output in case it is different from the default.



Note: Sample function is implemented in `nvsipl_camera -> CNvSIPLMaster.hpp`

- Application calls `INvSIPLCamera::GetImageAttrs(NvSciBufAttrs)`, which checks and validates the buffer attributes in case you chose to override the default format. Otherwise, Application adds attributes for the default formats.



Note: NVIDIA maintains the name of the API due to ABI compliance requirements. However, a name such as `VerifyandSetImageAttrsthe` is more representative of the functionality.

- Application calls `NvSciBufAttrReconcile` to reconcile all buffer attributes across other buffer attribute lists (other consumers for this buffer, which could be any downstream engine, such as VIC, IEP, and CUDA)
- Application calls `NvSciBufObjAlloc` on the reconciled attribute list to create a buffer object
- Application calls `INvSIPLCamera::RegisterImages(NvSciBufObj)`, which only accepts the buffer if one of the attribute lists used to reconcile and allocate was verified by SIPL using `INvSIPLCamera::GetImageAttrs`



Note:

- ISP output images will be a binary match if the default NITO provided by NVIDIA is used while requesting the same output formats with same resolution.
- The ISPO and ISP1 outputs can have different formats as long as they both are YUV type.
- The camera tuning does not have to be re-run for different formats since formats are just different ways to represent images in the memory. However, the data itself will differ as the representation of the memory varies across formats.

OverridelImageAttributes

ISP supports three types of formats:

- YUV Semi Planar Images
- YUV Packed Images—Luma is a type of YUV Packed Image
- RGBA Packed Images

Creating packed type buffers (2 and 3) are straightforward: you use `NvSciBufImageAttrKey_Plane***` type attributes to override Image formats.

For semi-planar type buffers (1), however, `NvSciBuf` provides attribute keys to create buffers of the above types using two methods:

- Using `NvSciBufImageAttrKey_Plane***` type attributes
- Using `NvSciBufImageAttrKey_Surf***` type attributes

NvSciBuf added the Surf type attributes to make the task of creating multi-plane image buffers easy for the user. However, in Plane type attributes you must provide the height and width of each plane to determine the memory layout of the image.

In SIPL the library oversees calculations for height and width of the buffer factoring in multiple criteria such as sensor output resolution, input crop, downscale, and so on. This is complicated if you supply Plane type attributes to SIPL while trying to create multi-plane images. Hence, SIPL restricts you to Surf type attributes when using multi-plane YUV images.

Examples

For YUV 444 SEMI-PLANAR UINT 16 BLOCK LINEAR image, the following attributes should be set to the corresponding values:

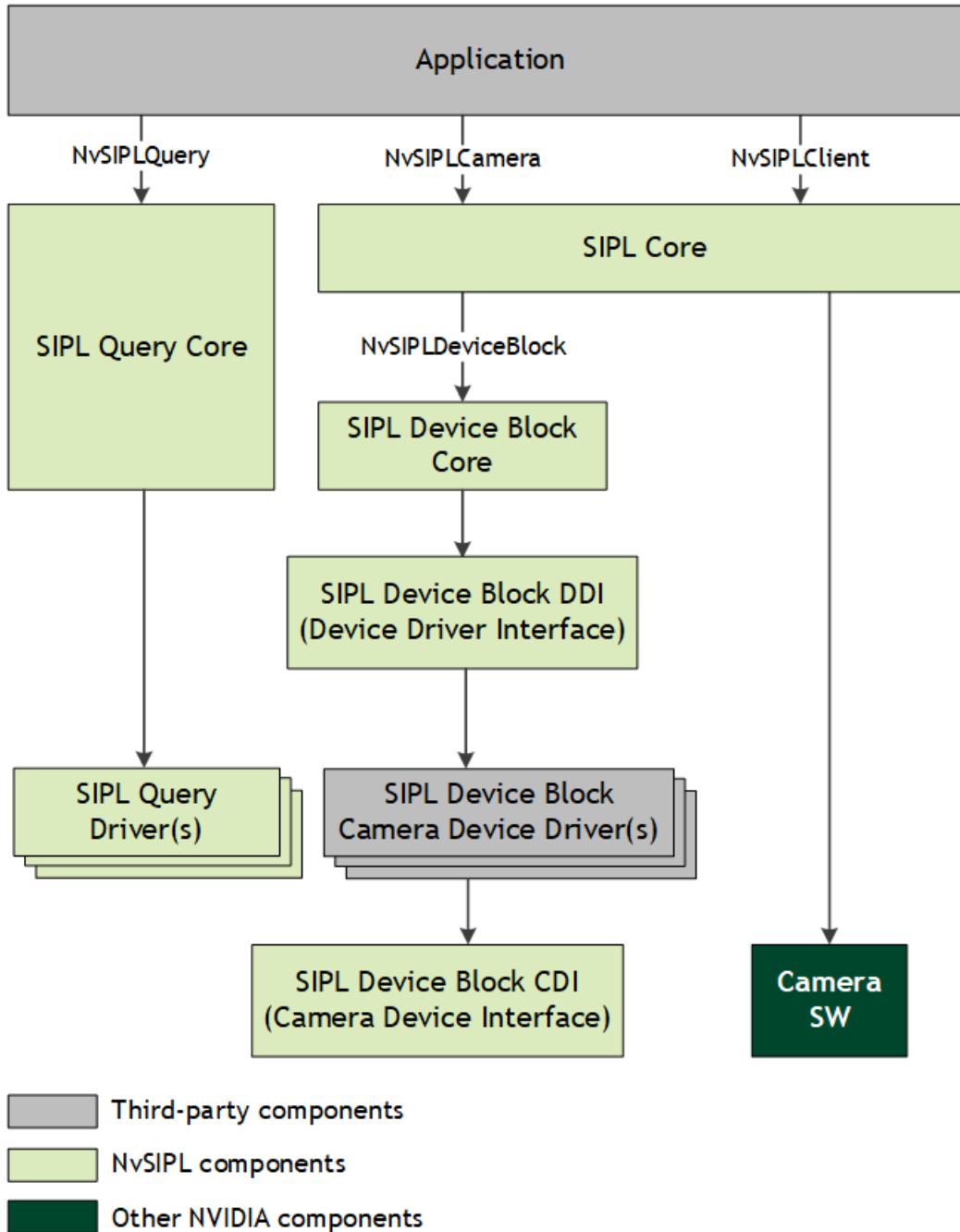
Attribute Name	Attribute Value
NvSciBufGeneralAttrKey_Types	NvSciBufType_Image
NvSciBuflImageAttrKey_SurfType	NvSciSurfType_YUV
NvSciBuflImageAttrKey_SurfBPC	NvSciSurfBPC_16
NvSciBuflImageAttrKey_SurfMemLayout	NvSciSurfMemLayout_SemiPlanar
NvSciBuflImageAttrKey_SurfSampleType	NvSciSurfSampleType_444
NvSciBuflImageAttrKey_SurfComponentOrder	NvSciSurfComponentOrder_YUV
NvSciBuflImageAttrKey_SurfColorStd	NvSciColorStd_REC709_ER
NvSciBuflImageAttrKey_Layout	NvSciBuflImage_BlockLinearType

For YUV 444 PACKED UINT 8 BLOCK LINEAR image, the following attributes should be set to the corresponding values

Attribute Name	Attribute Value
NvSciBufGeneralAttrKey_Types	NvSciBufType_Image
NvSciBuflImageAttrKey_PlaneCount	1
NvSciBuflImageAttrKey_Layout	NvSciBuflImage_BlockLinearType
NvSciBuflImageAttrKey_PlaneColorFormat	NvSciColor_A8Y8U8V8
NvSciBuflImageAttrKey_PlaneColorStd	NvSciColorStd_REC709_ER

6.7.4.3 SIPL Architecture

The following diagram illustrates the architecture of the SIPL framework.



The SIPL Query component is used to query data about which external devices are supported, how they are connected to the platform, and how they should be configured. It returns this information in the form of a PlatformCfg struct.

The SIPL Device Block component initializes and controls the external devices attached to the platform. SIPL Device Block is composed of four primary sub-components: Core, Device Driver Interface (DDI), Camera Device Drivers, and Camera Device Interface (CDI). As a combined component, SIPL Device Block uses I2C to program the settings specified in the PlatformCfg struct. Additionally, it initializes the GMSL deserializers, brings up the serializer/deserializer (serdes) link(s) between the camera modules and the deserializers,

and begins streaming from the sensors on the camera modules. Please note that the application does not directly use SIPL Device Block to control and interact with external devices; instead, it must go through SIPL Core.

The SIPL Core component orchestrates the entire capture process from initialization, through running, to deinitialization. It uses the SIPL Device Block component to initialize the external devices, and initializes the SoC to process captured images. It also allows the user to request images both before and after they have gone through the ISP pipeline and provides queues to deliver those images to the user.

6.7.4.4 SIPL Use Cases

SIPL supports special use cases, such as receive-only mode and a custom auto control plugin.

The sub-sections provide detailed information for the following usecases:

- Camera SIPL Receive-Only Mode
- Registering a User Defined Auto Control Plugin

6.7.4.4.1 Camera SIPL Receive-Only Mode

Receive-Only Mode is an operation mode of the Camera SIPL library, on a secondary NVIDIA® Tegra host, to receive and process MIPI CSI frames without interacting with the camera module hardware. The client on the primary host perform control and operation of the camera module hardware. Frame capture settings should be configured identically on the secondary receive only client for compatibility.

Use case Examples

- Data Recording
- Load balancing, such as running different networks on the same inputs concurrently
- Allowing data to cross safety boundaries:
 - Primary ASIL safety host is the sole controller of camera hardware on a particular camera hardware block
 - Secondary host may be QM or a lower ASIL level, such as a back-up camera, data recorder, or infotainment

Receive-Only Mode Operation

Receive-only mode is enabled by setting the `isPassiveModeEnabled` flag in the device block platform configuration. The customer application should ensure that the SIPL platform configuration on both SoCs are mirrored identically, or are otherwise compatible for capture, and then synchronize the initialization and start of the primary client before the secondary client.

- APIs with I2C accesses and power control calls to deserializers and camera modules, such as sensors, serializers, and EEPROMs, are rendered NOP, and they return a success status immediately. The exceptions are that I2C writes are still performed in the deserializer driver at `Init()` and `Deinit()` to enable and disable replication mode on the deserializer hardware respectively, and to run CSI deskew in D-Phy mode only.

The NVCSI pad on the receiving Tegra must be enabled when the deserializer drives CSI output to the lanes. Otherwise, permanent damage to the chip is possible. It is not possible to enable replication mode on the deserializer from the primary SoC before the secondary SoC client is initialized.

- External camera hardware interrupts continue to be monitored and propagated to the secondary client as usual if physically connected and configured. These signals may be shared and jointly delivered with the primary client if it is also physically connected and configured.

External Camera Hardware

Camera hardware device errors and interrupts may be propagated to both the primary and secondary SoCs. Error interrupt lines are connected to both SoCs on reference Tegra boards, such as the NVIDIA Orin Devkit. SIPL propagates error interrupt notifications to the client regardless of whether the receive-only mode flag is set in the platform configuration. The secondary mode client is typically unable to respond such errors; it may be forbidden from performing I2C communication to retrieve the detailed error status and effect a link recovery procedure. This indication of hardware fault may be useful to arrest a capture pipeline for a mission-critical application on the secondary SoC in the event of latent source corruption.

The client system architecture implements the inter-Tegra error notification delivery and handling strategy, which coordinates the responses, possibly synchronized, of the independent clients.

Tegra Capture and Processing

Capture and processing errors originating from the secondary SoC are propagated to the client through SIPL and other mechanisms, such as 3LSS, as usual and without change.

Link Recovery

The procedure to perform link recovery in response to error notifications can occur on the primary SoC client, but this will usually interfere with the secondary SoC capture.

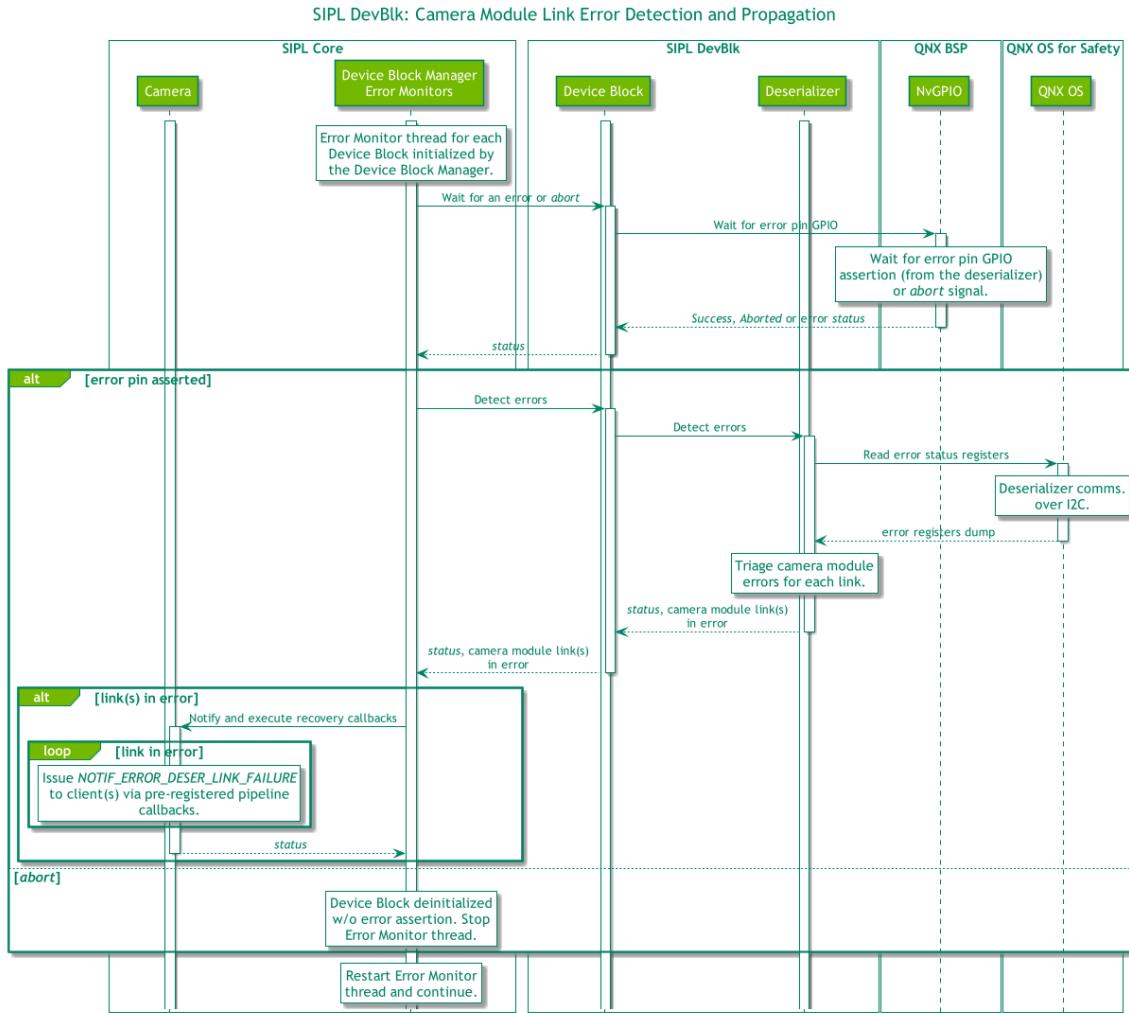
Link recovery is currently not supported on the secondary SoC client. A re-launch of the non-safety receive-only mode SIPL client may be necessary to achieve restoration of service.

	Drive OS 5.x (T19x)	Drive OS 6.0.3.0+ (T23x)
Linux (non-safety)	Yes	Yes
QNX (standard)	Yes	Yes
QNX (safety)	No	Yes

The SIPL library contains a basic callback-based camera module link error notification and recovery system. Each deserializer on the Camera Interface Module (CIM) has an error GPIO pin to indicate the appearance of an error in its links. An error monitor thread for each device block dumps the deserializer error status registers, automatically

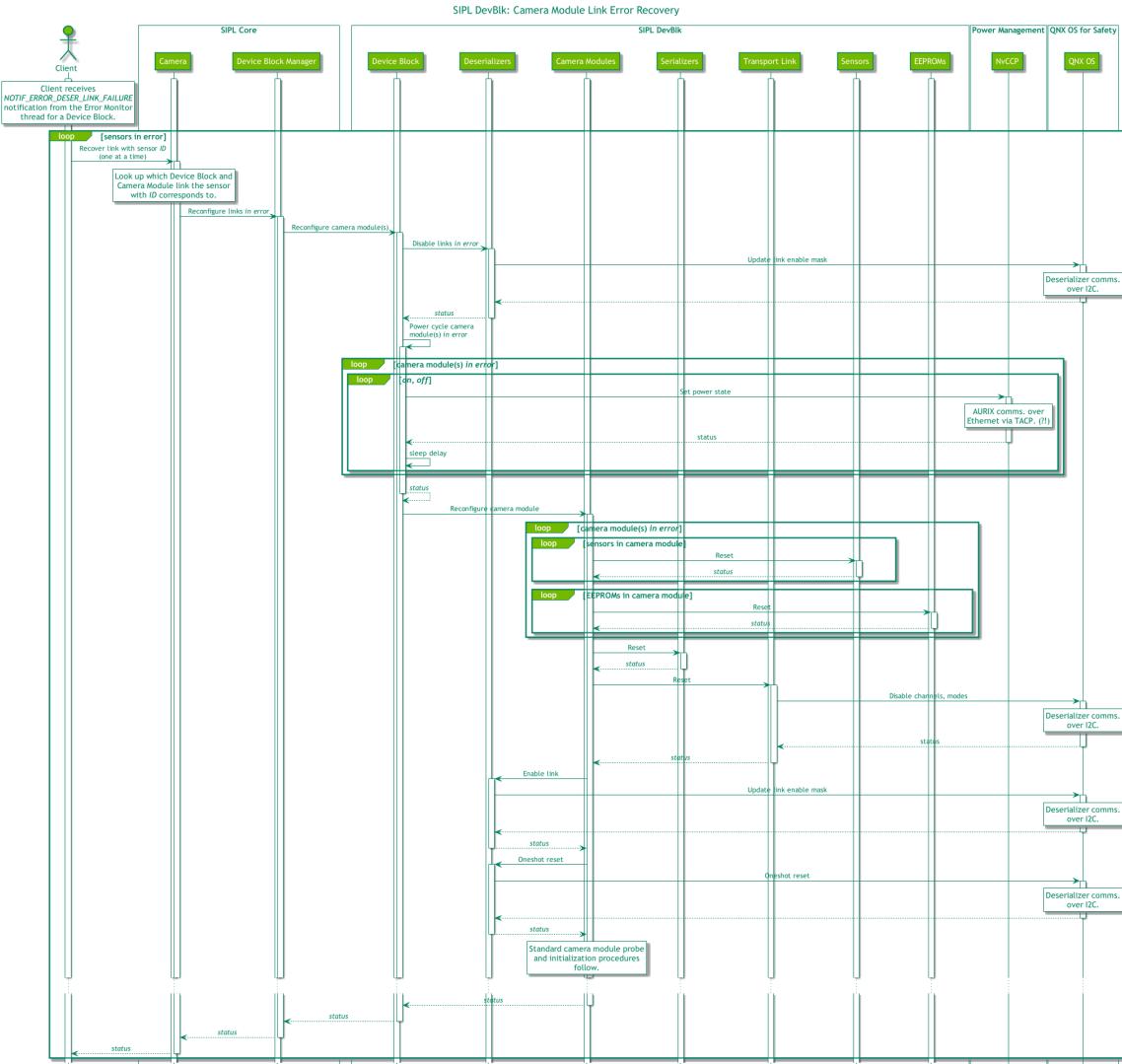
clears the error GPIO, and notifies the client via its pre-registered callback of a NOTIF_ERROR_DESER_LINK_FAILURE event.

A mask indicating which of the camera modules is in error is communicated to the client, and the deserializer error status registers contain more information about the specific error conditions.



Camera module error recovery is not automatic and triggered by the client. Recovery occurs at the link granularity. Individual camera modules and their links are independently disabled at the deserializer and each subdevice is re-initialized, as shown in the following diagram.

The camera pipeline, NVCSI/VI/ISP, is not reset by SIPL. The client coordinates frame error handling and resubmission of capture and process requests during recovery.



6.7.4.4.2 Registering a User Defined Auto Control Plugin

The application is responsible for allocating and deallocating the plugin object. To avoid resource allocation during safety critical operation, the plugin should only allocate required resources during initialization and deallocate during deinitialization.

To register a user-defined (custom) auto control plugin, the user must derive from the `ISiplControlAuto` interface class and implement its functions.

```
class ISiplControlAuto {
public:
    virtual SIPLStatus Process(const SiplControlAutoInputParam& inParams,
                               SiplControlAutoOutputParam& outParams) = 0;
    virtual SIPLStatus GetNoiseProfile(
        const SiplControlEmbedInfo& currFrameEmbedInfo,
        const uint32_t maxSupportedNoiseProfiles,
        uint32_t& noiseProfile) {
        noiseProfile = 0;
    }
};
```

```

        return NVS IPL_STATUS_OK;
    }
    virtual ~ISiplControlAuto() = default;
protected:
    ISiplControlAuto() = default;
private:
    ISiplControlAuto(const ISiplControlAuto&) = delete;
    ISiplControlAuto& operator= (const ISiplControlAuto&) = delete;
};

```

6.7.4.4.2.1 Class Public Functions

The public functions for `ISiplControlAuto` are as follows.

Member	Description
Process	The function is called for every frame by SIPL to process the input parameters (<code>SIPLControlAutoInputParam</code>) and generate the output settings (<code>SiplControlAutoOutputParam</code>) for future images.
GetNoiseProfile	The function is called for every frame by SIPL to generate the output noiseProfile based on the input parameters (<code>SiplControlEmbedData</code> and <code>maxSupportedNoiseProfiles</code>).
<code>~ISiplControlAuto()</code>	Default destructor.

Process Function

The Process function is called in the first frame without ISP Statistics or when the ISP has processed the captured image and generated the statistics.

The plugin must receive input information in the `SIPLControlAutoInputParam` structure and populate the results into the `SIPLControlAutoOutputParam` structure. `SIPLControlAutoInputParam` and `SIPLControlAutoOutputParam` structures are described in the [Process Function Input Parameters](#) and [Process Function Output Parameters](#) sections.

```

SIP LStatus Process(const SiplControlAutoInputParam& inParams,
                    SiplControlAutoOutputParam& outParams)

```

Parameters	<p>inParams: Specifies the input parameters to the plugin.</p> <p>outParams: Specifies the generated output parameters by the plugin.</p>
Return Value	<p>NVS IPL_STATUS_OK: Success.</p> <p>NVS IPL_STATUS_ERROR: An error occurred.</p>

Process Function Input Parameters

When the plugin Process function is called, it must process the input information and generate the output parameters.

The plugin supports the following input categories:

- > Sensor Attribute Properties
- > Embedded Data and line information
- > Histogram and LAC statistics
- > Flicker band statistics

SipControlAutoInputParam

The input information is passed in the following structure:

```
struct SipControlAutoInputParam {
    SipControlEmbedData      embedData;
    DevBlkISCSensorAttributes sensorAttr;
    SipControlIspStatsInfo   statsInfo;
};
```

Structure	Parameter
SipControlAutoInputParam	<ul style="list-style-type: none"> > embedData: Specifies the Embedded Settings of the frame. > sensorAttr: Specifies the attributes of the sensor. > statusInfo: Specifies the stats data.

SipControlEmbedData

The structure members provide embedded data for the image that is being processed. That information is a set of registers of the image sensor.

The definition is as follows:

```
struct SipControlEmbedData {
    SipControlEmbedInfo          embedInfo;
    DevBlkISCFrameSeqNum        frameSeqNum;
    DevBlkISCEmbeddedDataChunk  topEmbeddedData;
    DevBlkISCEmbeddedDataChunk  bottomEmbeddedData;
};
```

Structure	Parameter
SipControlEmbedData	<ul style="list-style-type: none"> > embedInfo: Holds the parsed embedded info for the captured frame. > frameSeqNum: Holds frame sequence number for the captured frame. > topEmbeddedData: Embedded buffer at the beginning of the frame. > bottomEmbeddedData: Embedded buffer at the end of the frame.

DevBlkISCSSensorAttributes

The structure members provide attributes data for the image sensor.

```
struct DevBlkISCSSensorAttributes {
    char sensorName[DEVBLK_ISC_MAX_SENSOR_NAME_LENGTH];
    uint32_t sensorCFA;
    char sensorFuseId[DEVBLK_ISC_MAX_FUSE_ID_LENGTH];
    uint8_t numActiveExposures;
    DevBlkISCAttrRange sensorExpRange[DEVBLK_ISC_MAX_EXPOSURES];
    DevBlkISCAttrRange sensorGainRange [...];
    DevBlkISCAttrRange sensorWhiteBalanceRange [...];
    float_t sensorGainFactor[DEVBLK_ISC_MAX_EXPOSURES];
    uint32_t numFrameReportBytes;
};
```

Structure	Parameter
DevBlkISCSSensorAttributes	<ul style="list-style-type: none"> > sensorName: Holds the name attribute. > sensorCFA: Holds the CFA attribute. > sensorFuseId: Holds the fuse ID attribute. > numActiveExposures: Holds the number of active exposures attribute. > sensorExpRange: Holds the sensor exposure ranges for active exposures. > sensorGainRange: Holds the sensor gain ranges for active exposures. > sensorWhiteBalanceRange: Holds the sensor white balance ranges for active exposures. > sensorGainFactor: Holds the additional sensor gain factor between active exposures. These gain factors describe the sensitivity difference between the exposures. > numFrameReportBytes: Holds the number of frame report bytes supported by the sensor.

SiplControlIspStatsInfo

This structure contains the statistics data.

```
struct SiplControlIspStatsInfo {
    const NvSiplISPLocalAvgClipStatsData* lacData[2];
    const NvSiplISPLocalAvgClipStats* lacSettings[2];
    const NvSiplISPHistogramStatsData* histData[2];
    const NvSiplISPHistogramStats* histSettings[2];
    const NvSiplISPFLickerBandStatsData* fbStatsData;
    const NvSiplISPFLickerBandStats* fbStatsSettings;
```

```
};
```

Structure	Parameter
SiplControlIspStatsInfo	<ul style="list-style-type: none"> > lacData: Holds const pointers to LAC stats data from LAC-0, LAC-1 blocks in ISP. > lacSettings: Holds const pointers to LAC stats settings from LAC-0, LAC-1 blocks in ISP. > histData: Holds const pointers to Histogram stats data from HIST-0, HIST-1 blocks in ISP. > histSettings: Holds const pointers to Histogram stats settings from HIST-0, HIST-1 blocks in ISP. > fbStatsData: Holds const pointer to Flicker Band stats data from FB block in ISP. > fbStatsSettings: Holds const pointer to Flicker Band stats settings from FB block in ISP.

Process Function Output Parameters

When the plugin Process function is called, it must process the input information and generate the output parameters.

The plugin must generate the information in the following categories:

- > Exposure control
- > White balance control
- > Histogram statistics settings
- > LAC statistics settings
- > Flicker-band statistics settings
- > SiplControlAutoOutputParam

SiplControlAutoOutputParam

The `SiplControlAutoOutputParam` structure provides the ISP stats settings, exposure settings, and white balance settings calculated by plugin based on the ISP stats received in [Process Function Input Parameters](#).

The output is passed in the following structure:

```
struct SiplControlAutoOutputParam {
    SiplControlAutoSensorSetting    sensorSetting;
    SiplControlAutoAwbSetting      awbSetting;
    SiplControlIspStatsSetting     newStatsSetting;
    float_t                         ispDigitalGain;
```

```
};
```

Structure	Parameter
SipControlAutoOutputParam	<ul style="list-style-type: none"> > sensorSetting: Holds sensor exposure and gain settings. > awbSetting: Holds AWB settings. > newStatsSetting: Holds Stats Settings. > ispDigitalGain: Holds the digital gain to be applied in ISP.

SipControlAutoSensorSetting

This structure contains the sensor settings.

```
struct SipControlAutoSensorSetting {
    uint8_t numSensorContexts;
    DevBlkISCExposure exposureControl[...];
    DevBlkISCWhiteBalance wbControl[...];
};
```

Structure	Parameter
SipControlAutoSensorSetting	<ul style="list-style-type: none"> > numSensorContexts: Holds the number of sensor contexts to activate. > exposureControl: Holds the sensor exposure settings to set for each context. > wbControl: Holds the sensor white balance settings to set for each context.

SipControlAutoAwbSetting

This structure contains the AWB, CCT and CCM settings.

```
struct SipControlAutoAwbSetting {
    SipControlAutoAwbGain wbGainTotal[...];
    float_t cct;
    float_t ccmMatrix[...][...];
}
```

Structure	Parameter
SipControlAutoAwbSetting	<ul style="list-style-type: none"> > wbGainTotal: Total white balance gains, including both sensor channel gains and ISP gains. > cct: Correlated Color Temperature. > ccmMatrix: Color Correlation Matrix.

SipControlIspStatsSetting

This structure contains the stat block settings for the ISP.

```
struct SipControlIspStatsSetting {
```

```

    bool valid;
    NvSiplISPLocalAvgClipStats lac[2];
    NvSiplISPHistogramStats hist1;
    NvSiplISPFLickerBandStats fbStats;
}

```

Structure	Parameter
SiplControlSpStatsSetting	<ul style="list-style-type: none"> > valid Settings: to control ISP stats blocks are valid or not. > Lac: Settings for 2 LAC stats ISP blocks. > hist1: Settings for Histogram 1 stats blocks. > fbStats: Settings for Flicker Band stats block.

GetNoiseProfile Function

The GetNoiseProfile function receives the embedded information for the current frame in `SiplControlEmbedInfo` structure and the parameter `maxSupportedNoiseProfiles` variable and puts the generated noise profile number in `noiseProfile` variable. `SiplControlEmbedInfo` structure is defined in the [SiplControlEmbedData](#) section and `maxSupportedNoiseProfiles` is an integer ranging from 1 to 32, specifying the max number of allowed noise profiles to be used by the plugin.

Declaration

```

SIPLStatus GetNoiseProfile(const SiplControlEmbedInfo& currFrameEmbedInfo, const uint32_t
    maxSupportedNoiseProfiles,
    uint32_t& noiseProfile)

```

Parameters	currFrameEmbedInfo: Specifies the input embedded info parameters for the current frame. maxSupportedNoiseProfiles: Specifies the input max number of allowed noise profiles to be used by the plugin. noiseProfile: Specifies the output calculated noise profile number.
Return Value	NVIPL_STATUS_OK: Success. NVIPL_STATUS_ERROR: An error occurred.

6.7.4.4.2.2 GetNoiseProfile Function Input Parameters

When the GetNoiseProfile function is called, it must process the following input information and generate the output parameter.

- > `SiplControlEmbedData`
- > `maxSupportedNoiseProfiles`

SiplControlEmbedData

This structure contains the embedded data for the image that is being processed. This structure is explained in the [SiplControlEmbedData](#) section.

maxSupportedNoiseProfiles

This parameter is an integer specifying the max number of allowed noise profiles to be used in plugin.

6.7.4.4.2.3 GetNoiseProfile Function Output Parameters

The output of `GetNoiseProfile` function is the generated noise profile number, `noiseProfile`. The valid range for the output `noiseProfile` is:

```
0<= noiseProfile < maxSupportedNoiseProfiles.
```

6.7.4.4.2.4 Plugin Input Data Range Requirements

The plugin must check the range for all the input parameters based on the valid ranges provided in the NVIDIA DRIVE OS SDK API Reference Documentation and return an error if any value is not in range. In addition, the plugin must return an error if `expTimeValid` or `gainValid` is false in `EmbedInfo`.

6.7.4.4.2.5 Plugin Output Data Range Requirements

The plugin must output values within the ranges provided in the NVIDIA DRIVE OS SDK API Reference Documentation. In addition, SIPL code would detect the following situations and report an error:

- > When the product of output digital gain and any of the output `wbGainTotal` gains is greater than 8 or less than 0.
- > For each frame, SIPL calculates the total exposure as a product of exposure and gain as follows:

```
Total_Exposure = exposureTime[0] * sensorGain[0]
```

Only the 0th exposure and gain values (corresponding to the long exposure frame) are used for this calculation. In order to prevent abrupt change between consecutive frames, SIPL calculates the ratio of *Total_Exposure* between Frame N and N+1 as follows:

```
Exposure_Ratio = Total_Exposure[N+1] / Total_Exposure[N]
```

- *Exposure_Ratio* must be within the range of [1/1024.0, 1024.0].

- > When any coefficient in product of output `ccmMatrix` and inverse of the Color Space Conversion (CSC) matrix is greater than 8 or less than -8.

The default `CSC_Inv` matrix is as follows:

```
CSC_Inv[0] = {0.2126, -0.11457, 0.5};  
CSC_Inv[1] = {0.7152, -0.38543, -0.45415};  
CSC_Inv[2] = {0.0722, 0.5, -0.04585};
```

In addition, fusion in ISP is not supported. Therefore, the output `SiplControlAutoAwbGain` must be filled with valid values for the first plane (`SiplControlAutoAwbGain[0]`).

6.7.4.4.2.6 ISP Processing and 24-bit Sensor Output

NVIDIA recommends the following programming to process 24-bit sensors: The exposure combination is done in the sensor to generate the 24-bit linear pixel values. Following the exposure combination, the sensor performs a PWL compression on the 24-bit linear values to generate the 12-bit compressed pixel values.

The PWL compression on 24-bit values in the sensor is such that when ISP applies a PWL decompression to retrieve 20-bit data, the 20-bit data are effectively the result of 24-bit linear data compressed by a power curve. The power curve is designed to preserve the dynamic range of the sensor.

All the blocks following the PWL decompression (linearization) block in ISP (specifically the statistics data: LAC0, LAC1, HIST0, HIST1, and FB) operate on the 20-bit compressed data.

Therefore, when using the statistics data, you can write software to linearize the data by using the inverse power-curve.

The semi-raw RGBFP16 output generated by ISP is also compressed using this power-curve. Obtain this linear data by writing the software to use the inverse power-curve.

6.7.4.5 Camera Authentication

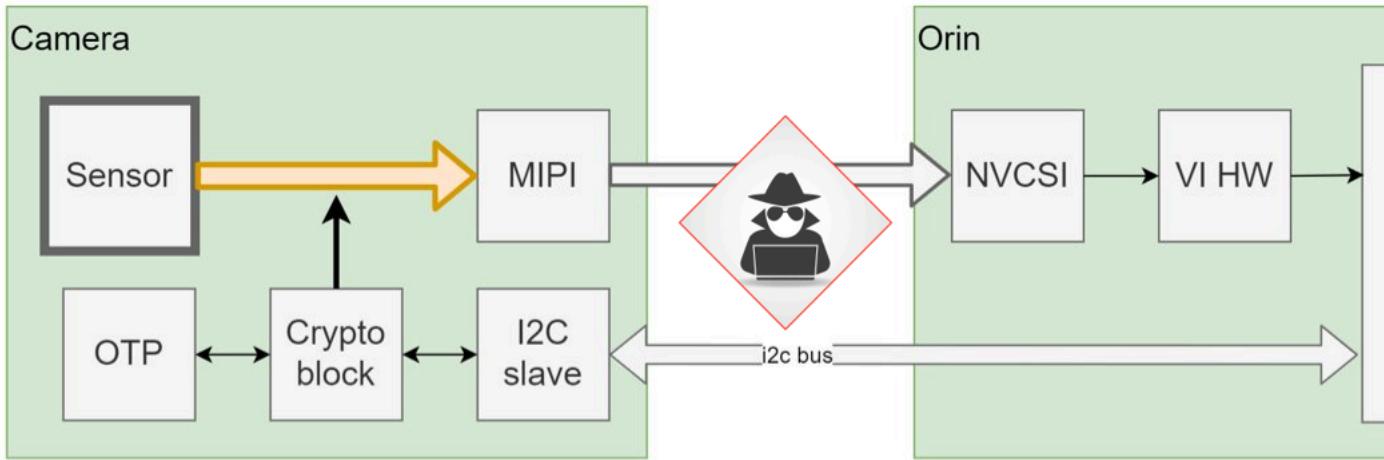
NVIDIA DRIVE® OS 6 introduces features to authenticate camera sensors, the control messages between camera sensors and Drive OS, and the image frames received from the camera sensors. This topic describes the following:

- > Camera authentication features and supported cameras.
- > How security features are deployed and configured with NVIDIA Drive OS 6 and how applications can use these features.
- > How authentication features can be enabled for additional camera sensors.

The target audience of this topic consists of OEMs and system integrators, software developers writing applications for Drive OS 6, and camera device driver writers.

Motivation and Use Cases

Cameras are used in autonomous vehicles for perception of the environment surrounding the vehicle. Camera data is typically used to assist the driver in maneuvering the vehicle, or to control the vehicle direction and speed, depending on the level of autonomy.



Several security threats are possible and applicable to this architecture. At a high level they are:

- > **Tampering of the camera or its sensor:** An entity that gains physical access to the camera and its sensor can tamper with the image or other data sent by the camera.
- > **Tampering of image data:** An entity that gains physical access to hardware CSI image data read channels can tamper with image data. Image data can be tampered in a way to disrupt the perception application that consumes this data. This can result in misclassification of image data.
- > **Tampering of I2C Settings:** An entity that gains physical access to HW I2C write channels can tamper with settings written by sensor drivers. These settings control the camera and can cause the quality of images sent by the camera to be affected to the point where images may not be correctly processed by perception applications.
- > **Image Frame Replay and Forgery:** An entity that gains physical access to hardware CSI image data read channels can intercept and replay previously sent image frames or forge new image frames. Image data can be tampered with to disrupt the perception application that consumes this data. This can result in misclassification of image data.

The camera authentication features provide mitigations against these threats.

Camera Authentication Features

NVIDIA DRIVE OS provides the following camera authentication features to mitigate against threats.

- > Camera authentication: establish the authenticity of the camera module and its sensor.
- > Image authentication: ensure that the image data is coming from the expected sensor, has not been tampered with, and is not a replay of a previously sent camera frame.
- > Control authentication: ensure that control messages (typically exchanged with i2c) are not tampered with.

Supported Products

There is no established standard for authenticating camera modules or authenticating data from camera modules. The camera authentication algorithms and protocols used depend on the capabilities provided by a specific camera model, with each camera vendor implementing their own proprietary solution.

6.7.4.5.1 Platform Camera Configuration

NVIDIA DRIVE® OS Safety build enforces authentication of all camera modules that are enabled in platform camera topology configuration. There is no option to disable the authentication step in a Safety build. DRIVE OS SIPL core refuses an application request to start a camera streaming for any module whose authentication cannot be successfully finished for whatever reason:

- > SIPL device driver for a sensor does not support authentication
- > Physical camera module does not support authentication
- > Physical camera module is not provisioned with required key materials
- > Authentication is supported, but fails during session establishment for any reason

For more information, refer to [Understanding the Sensor Input Processing Library \(SIPL\) Framework](#)

Non-safety and safety debug builds have an option to control authentication for a specific camera module through SIPL camera platform configuration. The SIPL camera core framework attempts to authenticate camera modules that have the `isAuthEnabled` flag set to `true` in a platform config. Following is an example SIPL database JSON file description of the IMX728 module:

```
{
  "cameraModules": [
    {
      "name": "V1SIM728S2RU2030NB20",
      "description": "IMX728 RGGB module - 30FOV - MIPI-IMX728, MAX96717",
      "serializer": {
        "name": "MAX96717F"
      },
      "sensors": [
        {
          "name": "IMX728",
          "i2cAddress": "0x1C",
          "isAuthEnabled": true,
          "virtualChannels": [
            {
              "cfa": "rggb",
              "width": 3840,
              "height": 2160
            }
          ]
        }
      ],
      "eeproms": [
        {

```

```

        "name": "M24C04",
        "i2cAddress": "0x54"
    }
}
]
}
}

```

Platform integrators can enable authentication individually for each camera module in SIPL platform configuration for non-safety or debug builds. For more information, refer to [Adding a Sensor Configuration](#)

6.7.4.5.2 Application Support for Camera Authentication

This section provides information for camera application developers to activate camera authentication, receive, and process notification-related events (including failures).

The NVIDIA DRIVE OS SDK provides applications with a camera management interface through SIPL APIs. For additional information, refer to [Understanding the Sensor Input Processing Library \(SIPL\) Framework](#).

The SIPL framework manages all aspects of camera sensor authentication and provides status to an application using SIPL notifications.

6.7.4.5.2.1 Activating Authentication for a Camera Module

In a Safety DRIVE OS build, authentication is automatically enforced for all camera modules, without any control from application. DRIVE OS Camera core will fail to start if authentication fails for any active camera module.

For non-safety and DEBUG builds, application retrieves the “authentication enable” flag for each individual module by querying platform config. For details, refer to [SIPL Query](#). Platform config can be changed to control per-module authentication flag.

If the authentication flag is not explicitly set to False in a platform configuration, a non-safety build will use True value for the flag by default.

6.7.4.5.2.2 Module Initialization

The SIPL framework automatically runs the authentication step during the SIPL initialization call, `INvSIPLCamera::Init()`, when required by platform configuration (platform config is set by the `INvSIPLCamera::SetPlatformCfg()` call before `Init`). If the authentication step fails, the `INvSIPLCamera::Init()` method returns an error and Camera module initialization will not occur. Streaming can not be started for the module with the `NvSIPLCamera::Start()` call.

To identify the exact cause of failure, refer to the system logs.

6.7.4.5.2.3 I2C Transactions Authentication

NVIDIA DRIVE OS SIPL camera software communicates with camera sensors over i2c during Init and Runtime (streaming) steps:

- > At init time, default settings are pushed to a sensor over i2c bus before streaming can start.

- At runtime, image quality autocontrol algorithm in SIPL core can periodically decide to change sensor settings over i2c (exposure, white balance, and so on) based on changing lighting conditions.

Initialization time i2c transactions are performed when application calls the `INvSIPLCamera::Init()` method. Any failure to authenticate i2c transactions during Init stage is seen by application as a failure of `INvSIPLCamera::Init()` call, and streaming will not start.

At runtime (after camera streaming starts) SIPL manages optimal camera sensor settings internally by controlling sensor via i2c bus, transparently to camera user application. If i2c transactions fail, application is notified through SIPL notification queue. Refer to [nvSipl::NvSIPLPipelineNotifier::NotificationType](#) by `NOTIF_ERROR_CDI_SET_SENSOR_CTRL_FAILURE` event. I2C authentication failure is treated like any other type of I2C transaction error and is reported by the same event.

Application monitors the SIPL notification queue for `NOTIF_ERROR_CDI_SET_SENSOR_CTRL_FAILURE` events and performs actions required to recover the camera link when failure occurs.

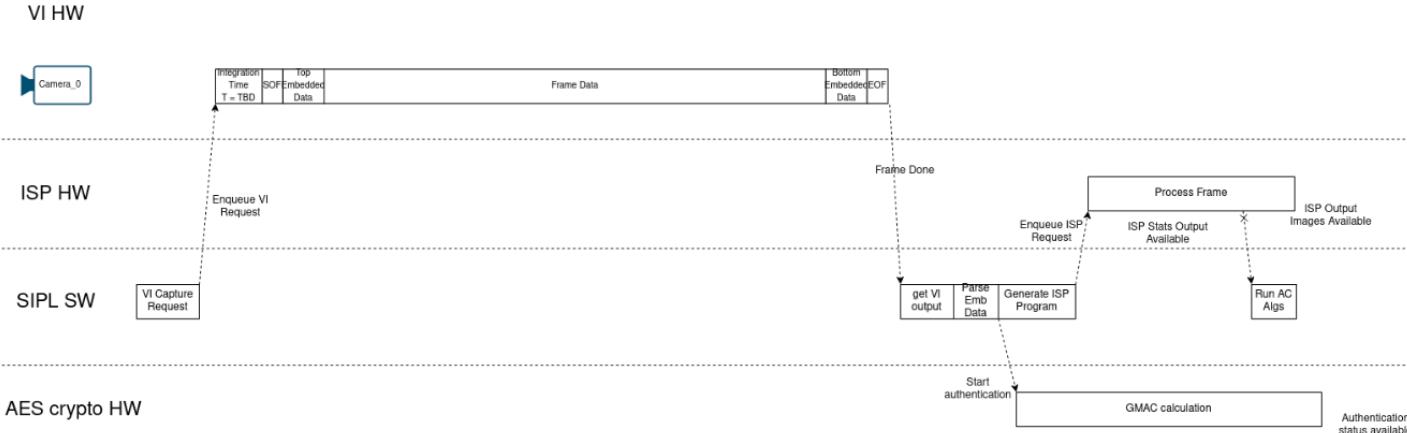
6.7.4.5.2.4 Image Authentication Status

Image authentication is performed by SIPL on RAW frame data after reception from CSI bus, unpacking (adding pad bits) and placing unpacked image data into system memory. Authentication runs on data before post-processing is applied.

The input pixel format depends on camera module configuration specified in camera platform config [VirtualChannelInfo](#) member `inputFormat`. Authentication can start immediately after frame is placed into memory in parallel with the image post-processing step.

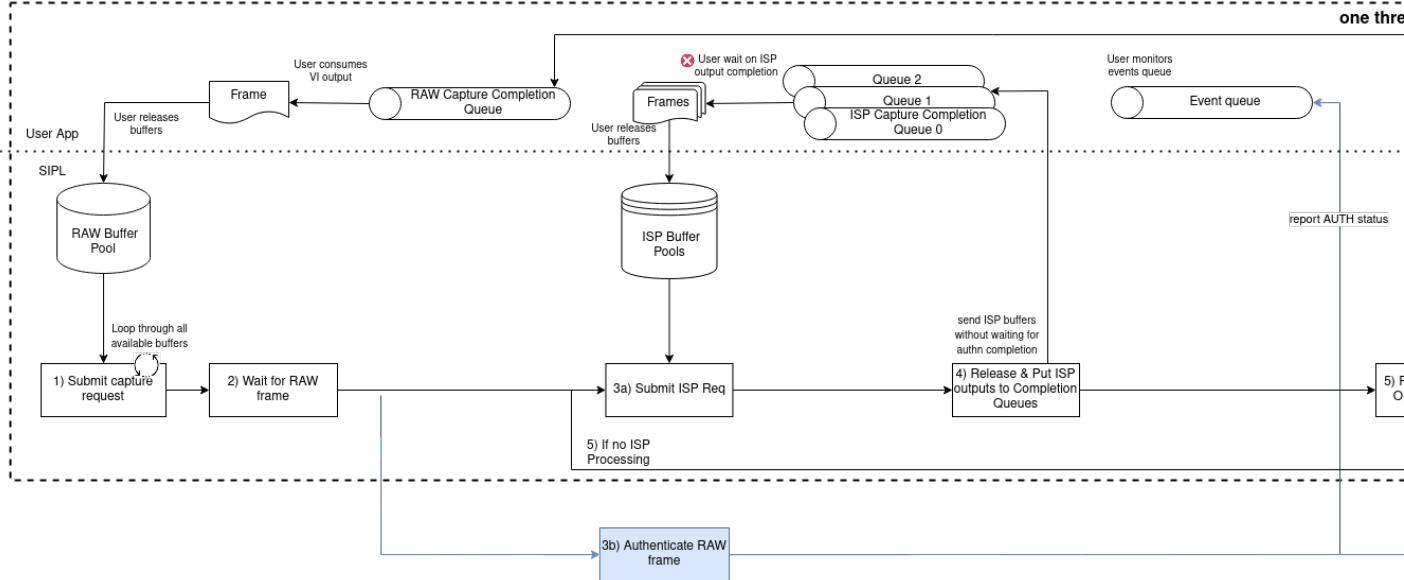
Application consuming post-processed frames from SIPL must wait for image authentication status before it makes a critical decision based on image pixel data. Application can use an image, for example sensor fusion, while it waits for authentication status, but it cannot make vehicle steering decisions until authentication status is available.

Authentication status of a frame is reported to Application separately from a frame itself, to reduce overall image processing latency, because authentication can be performed in parallel to post-processing of a RAW frame.



In case image authentication is done sequentially to post-processing extra latency would be introduced to camera capture pipeline. To avoid this SIPL performs authentication in parallel to post-processing step. Therefore, ISP post-processed images may be passed UP to a camera application before authentication is finished. Exact timing depends on the size of a frame to authenticate. It is then up to the application to use frame data and then wait for authentication status before safety decision is made (e.g., before vehicle steering decision is made).

This is illustrated on a following diagram showing standard SIPL image processing pipeline with an addition of Image Authentication Status reporting.



Asynchronous notification of authentication status for each frame is done via existing SIPL notification queue [INvSIPLNotificationQueue](#). Assumption is on camera application that it will monitor the notification queue for authentication status events and will do a matching between an event and previously received post-processed buffer.

SIPL notification queue events related to image authentication are as follows:

```
enum NotificationType {
```

```

...
NOTIF_ERROR_ICP_AUTH_FAILURE,
NOTIF_INFO_ICP_AUTH_SUCCESS,
...
}

NvSIPLPipelineNotifier::NotificationData {
    eNotifType type; // NOTIF_ERROR_ICP_AUTH_FAILURE or NOTIF_INFO_ICP_AUTH_SUCCESS
    uint32_t pipelineId;
    uint64_t frameSequenceNumber;
}

```

Application can identify each individual frame based on pipeline ID (one per Sensor) and a frame sequence number (always incrementing).

6.7.4.6 Using the SIPL Framework

6.7.4.6.1 Step 1: Querying Platform Configuration

First, use the SIPL Query component through the `INvSIPLQuery` interface to get a list of supported platform configurations. A **platform configuration** is structure (a struct) that lists properties of external devices and how they are connected to the platform. This includes but is not limited to:

- Device I2C addresses
- Image properties (frame rate, data type, image size, etc.)
- Deserializer MIPI properties (PHY mode, number of lanes, etc.)
- Which camera modules are connected to which CSI brick

This information is returned in the form of a list of `PlatformCfg` structs. Choose the `PlatformCfg` struct that reflects the current configuration. It is not strictly necessary to use the SIPL Query component, and the `PlatformCfg` struct may be obtained by other means, or even hard coded into the application.

6.7.4.6.2 Step 2: Initialize SIPL

To initialize SIPL Core, first use the `INvSIPLCamera::GetInstance` method to get an instance of a class that implements `INvSIPLCamera`. The following methods can then be used to configure the settings that SIPL Core is to use during capture:

- `INvSIPLCamera::SetPlatformCfg()`
- `INvSIPLCamera::SetPipelineCfg()`

Once SIPL Core has been configured, call `INvSIPLCamera::Init()` to initialize SIPL. This programs the external devices to prepare them to stream, receive, and process images. Note that this step prepares SIPL to capture and process images but does not actually start the process.

6.7.4.6.3 Step 3: Start SIPL

Next, register output buffers and configure autocontrol using the following methods:

- > `INvSIPLCamera::GetImageAttributes()`
- > `INvSIPLCamera::RegisterImages()`
- >
- > `INvSIPLCamera::RegisterAutoControlPlugin()`

All image buffers must be allocated with the attributes retrieved with the `INvSIPLCamera::GetImageAttributes()` function. Once allocated, all of the output buffers must then be registered with SIPL using the `INvSIPLCamera::RegisterImages()` function. Once registration of all output buffers is completed the pipeline can be started using `INvSIPLCamera::Start()`. This method programs the externally connected devices to start streaming images. After this method is called, capture is active. SIPL starts receiving images, processing them through ISP, and sending them to the image output queues.

6.7.4.7 Components

6.7.4.7.1 SIPL Query

The SIPL Query component implements the `INvSIPLQuery` interface and is contained in the shared library `libnvsipl_query.so`. It maintains a database of supported external image devices and their properties. It also maintains a database of platform configurations. The database contains the following types of objects:

Database Object Type	Description
<code>SensorInfo</code>	An object that stores the settings for an image sensor external device.
<code>EEPROMInfo</code>	An object that stores the settings for an EEPROM external device.
<code>SerInfo</code>	An object that stores the settings for a GMSL serializer external device.

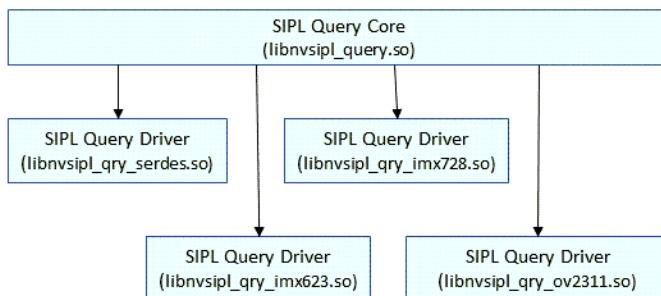
Database Object Type	Description
CameraModuleInfo	An object that stores the properties for a camera module external device. This consists of one <code>SerInfo</code> object, zero or more <code>EEPROMInfo</code> objects, and zero or more <code>SensorInfo</code> objects.
DeserInfo	An object that stores the settings for a GMSL deserializer external device.
PlatformCfg	An object that describes all external devices that are attached to the CSI ports of the SoC. This includes which <code>DeserInfo</code> objects are to be used for each camera group, and which <code>CameraModuleInfo</code> objects are to be used.

SIPL Query provides API calls to:

- Parse the default database
- Get information about all devices supported by the library
- Get a list of all supported platform configurations
- Retrieve a specific platform configuration by name
- Parse a user-provided `platform_config.json` file to override the `PlatformCfg` structs in the SIPL Query database
- Apply a mask to enable only specific deserializer links in a specific platform configuration

6.7.4.7.1.1 SIPL Query Drivers

SIPL Query creates the database at runtime by searching for **SIPL Query Drivers**. A Query Driver is a shared library in the form `libnvspipl_qry_*.so` that contains a collection of database objects. SIPL Query searches the paths provided in the `$LD_LIBRARY_PATH` environment variable for the first directory named `nvspipl_drv` and opens all Query Drivers in it.

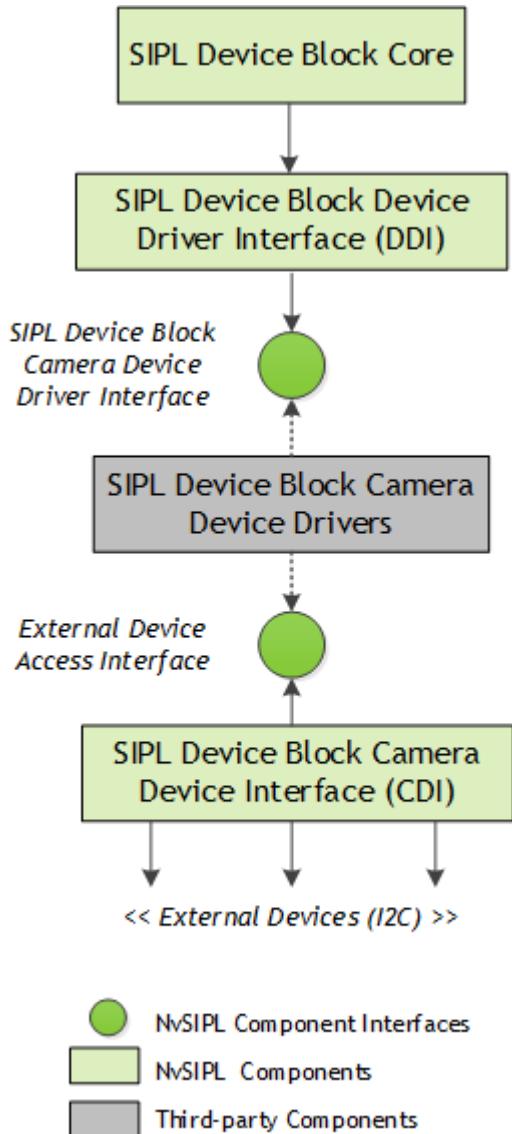


6.7.4.7.2 SIPL Device Block

SIPL Device Block is composed of four primary sub-components:

1. SIPL Device Block Core
2. SIPL Device Block Device Driver Interface (DDI)
3. (Third-party) SIPL Device Block Camera Device Drivers
4. SIPL Device Block Camera Device Interface (CDI)

The sub-components will be discussed in more detail below.



6.7.4.7.2.1 SIPL Device Block Core

Device Block Core provides a unified interface over arbitrary external device configurations to upper layers of the SIPL stack. To accomplish this and to accommodate custom silicon or driver solutions, Device Block Core defines supported external device types as a series of abstract interfaces. Device Block Core exclusively

deals with these interfaces in providing its main functionalities of hardware configuration and initialization, stream manipulation, error retrieval, etc.

Third-party **Device Block Camera Device Drivers** may provide implementations for the interfaces exported by Device Block Core allowing the SIPL stack to handle a wide variety of hardware solutions.

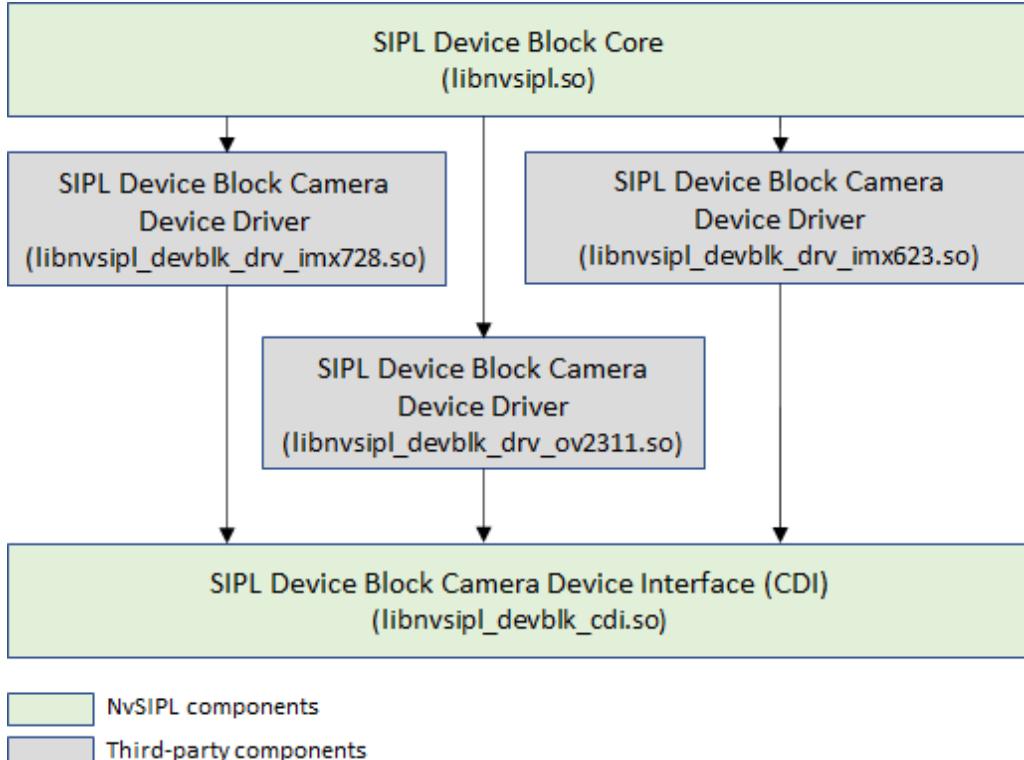
6.7.4.7.2.2 SIPL Device Block Device Driver Interface (DDI)

The Device Block Device Drivers Interface (DDI) sub-component is a set of C++ driver interface classes that 3rd-party vendors must inherit and implement in their camera device drivers. Using DDI, hardware-specific device drivers can be dynamically loaded into the client process by SIPL according to the provided platform configuration.

6.7.4.7.2.3 SIPL Device Block Camera Device Drivers

Device Block Core uses **Device Block Camera Device Drivers** to program external devices. A Device Block Camera Device Driver is a shared library in the form `libnvsipl_devblk_drv_*`.so that is used to program a specific type of camera module. Device Block Core selects which Device Block Camera Device Driver(s) to use based on the names of the camera modules that are attached to each GMSL link of the deserializer.

At runtime, Device Block Core searches the paths defined in the environment variable `$LD_LIBRARY_PATH` for the first directory named `nvsipl_drv` and opens all Device Block Camera Device Drivers in it. This allows you to add support for more devices by adding third-party Device Block Camera Device Drivers to the `nvsipl_drv` directory.



6.7.4.7.2.4 SIPL Device Block Camera Device Interface (CDI)

The Device Block Camera Device Interface (CDI) sub-component provides a common interface for Device Block Camera Device Drivers to access their hardware over an I2C interface. Additionally, CDI provides interfaces that expose various error signals from external devices.

6.7.4.7.3 SIPL Core

The SIPL Core component implements the `INvSIPLCamera` interface. It initializes the external devices through the SIPL Device Block component and initializes the platform to capture and process images.

SIPL Core also provides queues to contain the output buffers produced by the capture processes. You must read the buffers from these queues regularly to avoid starving the system of output buffers.

6.7.4.8 Retrieving NITO Metadata from a NITO File

The Fetch NITO Metadata API is a non-safety API that allows the user to retrieve NITO metadata from the parameter sets present in a NITO file. Depending on the workflow, NITO metadata can be helpful in use cases such as validation. One example can be confirming that the same NITO is used during initial frame capture and at a later stage (such as ISP processing).

A NITO file may contain one or more "parameter sets", and each parameter set contains "NITO metadata", defined as the parameter set ID, parameter set schema hash, and parameter set data hash unique to that parameter set. This API provides the ability to fetch this metadata for each parameter set in a NITO file.

The API consists of a single standalone function (see below), with an associated struct type and constants specific to a NITO file (described in the following section):

```
SIPLStatus GetNitoMetadataFromMemory(
    uint8_t const *const nitoMem,
    size_t const nitoMemLength,
    NvSIPLNitoMetadata *const metadataArray,
    size_t const metadataArrayLength,
    size_t *const metadataCount)
```

6.7.4.8.1 NvSIPLNitoMetadata Struct and Related Constants

The API provides constants that define the length of the Parameter Set ID and the Hash values. These are used by the API, and should be used when iterating over the retrieved metadata:

```
const size_t NITO_PARAMETER_SET_ID_SIZE = 16U
const size_t NITO_SCHEMA_HASH_SIZE = 32U
const size_t NITO_DATA_HASH_SIZE = 32U
```

NvSIPLNitoMetadata

This structure contains fixed-length arrays to store the metadata retrieved from a single parameter set.

```
struct NvSIPLNitoMetadata {
    uint8_t parameterSetID[NITO_PARAMETER_SET_ID_SIZE];
    uint8_t schemaHash[NITO_SCHEMA_HASH_SIZE];
    uint8_t dataHash[NITO_DATA_HASH_SIZE];
}
```

Structure	Parameter
NvSIPLNitoMetadata	<ul style="list-style-type: none"> > parameterSetID: Identifier of the parameter set. > schemaHash: Hash value of the parameter set schema. > dataHash: Hash value of parameter values.

6.7.4.8.2 GetNitoMetadataFromMemory Function

The `GetNitoMetadataFromMemory` function receives information about the NITO file in `nitoMem` and `nitoMemLength`, which is a pointer to a memory buffer loaded with the NITO file contents and an integer indicating the size of the NITO memory buffer, respectively. Once metadata is retrieved, it copies the data for each parameter set into `metadataArray`, an array of `NvSIPLNitoMetadata` struct (defined in the [NvSIPLNitoMetadata and Related Constants](#) section above). `metadataArrayLength` is an integer which represents length of this array (for example, the maximum number of elements that can be stored), used to guarantee there is enough storage space for all retrieved metadata tuples (ID, schema hash, data hash). The function also updates the `metadataCount` variable to indicate the number of metadata tuples retrieved.

Declaration

```
SIPLStatus GetNitoMetadataFromMemory(
    uint8_t const *const nitoMem,
    size_t const nitoMemLength,
    NvSIPLNitoMetadata *const metadataArray,
    size_t const metadataArrayLength,
```

```
size_t *const metadataCount)
```

Parameters	<ul style="list-style-type: none"> > nitoMem: Specifies the input pointer to the memory containing the NITO buffer. > nitoMemLength: Specifies the length (in bytes) of the input NITO buffer. The buffer length must be between 1 byte and 6MB (maximum NITO file size). > metadataArray: Specifies the output pointer to the NvSIPLNitoMetadata struct array that will store the retrieved metadata. > metadataArrayLength: Specifies the maximum number of elements that can be stored in the NvSIPLNitoMetadata struct array. > metadataCount: Specifies the output pointer to a location to store how many metadata tuples (ID, Schema Hash, Data Hash) were retrieved by the operation.
Return Value	<p>NVSIPL_STATUS_OK – Success</p> <p>NVSIPL_STATUS_BAD_ARGUMENT – Invalid parameters received, or more metadata tuples retrieved than maximum NvSIPLNitoMetadata struct array length indicated by metadataArrayLength.</p> <p>NVSIPL_STATUS_INVALID_STATE – Invalid state occurred</p> <p>NVSIPL_STATUS_OUT_OF_MEMORY – Memory allocation failed</p> <p>NVSIPL_STATUS_ERROR – Error occurred</p>

6.7.4.8.3 GetNitoMetadataFromMemory Function Input Parameters

The input parameters for the GetNitoMetadataFromMemory function are:

- > nitoMem - Pointer to the memory buffer loaded with NITO file contents.
- > nitoMemLength - Integer indicating memory buffer length
- > metadataArrayLength - Integer indicating length of output NvSIPLNitoMetadata struct array.

Input Parameter Restrictions

1. This function can only read NITO content from memory buffers. The user must provide functionality to copy NITO data from other containers (for example, files) into memory before using this API.
2. The length specified by nitoMemLength for the output NvSIPLNitoMetadata struct array must be greater than or equal to the number of parameter set(s) in the NITO buffer. The API will return the error NVSIPL_STATUS_BAD_ARGUMENT if the metadataArray cannot store all retrieved metadata tuples.
3. The API performs standard input parameter checks, such as null pointers and greater than 0 (zero) length.

6.7.4.8.4 GetNitoMetadataFromMemory Function Output Parameters

The output of `GetNitoMetadataFromMemory` function, using the NITO buffer and other information received as described in the [GetNitoMetadataFromMemory Function Input Parameters](#) section is the following:

- `metadataArray` (array of struct `NvSIPLNitoMetadata`), populated with metadata from all parameter set(s) in the NITO buffer.
- Updated `metadataCount` variable to indicate the number of metadata tuples retrieved.

6.7.4.8.5 API Usage

The following section describes the API workflow.

1. The user is required to instantiate and pass in five arguments:

- `nitoMem`: Pointer to location of memory that a NITO file is loaded into memory.



Note:

As mentioned in the section [Input Parameter Restrictions](#) earlier, NITO data in other containers (for example, files) must be copied into memory before using this API.

- `nitoMemLength`: Size of the memory pointed to that holds the NITO file.
- `metadataArray`: Pointer to a user-instantiated array of the `NvSIPLNitoMetadata` struct, which will be used to store the metadata retrieved.
- `metadataArrayLength`: Size of the metadata array, for example, the number of elements the array can hold.



Note:

As mentioned in the section [Input Parameter Restrictions](#) earlier, this must be greater than or equal to the number of parameter set(s) in the NITO buffer. The API will return an error if the `metadataArray` cannot store all retrieved metadata tuples.

- `metadataCount`: Pointer to a variable to store number of tuples in `metadataArray` on success.
2. On success, the function will populate the `NvSIPLNitoMetadata` struct array `metadataArray` with metadata from each parameter set and update the `metadataCount` variable to indicate the number of metadata tuples retrieved.
 - For printing the parameter set ID and hash values, the user can iterate over each element of the `metadataArray` using `metadataCount` and the NITO file constants described in [NvSIPLNitoMetadata and Related Constants](#) section.
 3. On failure, an error will be logged stating the possible reason and the corresponding error code.

Sample API usage code is provided in the `nvsipl_camera` application. In addition, command line arguments have been added to `nvsipl_camera` to allow users to retrieve metadata from the NITO files loaded by the application, as well as specify the number of parameter sets in the NITO file when necessary.

6.7.5 NvMedia Sample Applications

The NvMedia sample applications demonstrate how to use the NvMedia API to perform multimedia tasks. Multimedia based samples are built for single windowing compatibility that depends on the platform operating system.

NVIDIA DRIVE 6.0 Linux uses the X11 windowing system.

Graphics OpenGL ES samples are built and organized for compatibility with both Wayland and X11 windowing systems with subdirectories for both. For information on the graphics OpenGL ES samples, see [Building and Running Samples](#) in [Graphics Programming](#)

6.7.5.1 Building and Running the NvMedia Samples

The NvMedia samples installed on the Linux host development system are already compiled and ready to run. If you modify a sample, though, you must rebuild it on the host Linux system. This section explains how.

Additionally, before you can run a sample, you must define environment variables and optionally customize a configuration file.

6.7.5.1.1 Building the NvMedia Samples

Each sample comes with source code and a makefile. Use these procedures to build the desired sample.

6.7.5.1.1.1 To build a sample application

1. On the host system, enter the commands:

```
cd <top>/drive-linux/samples/nvmedia/<sample_dir>
make clean
make
```

Where <sample_dir> is the sample directory.

2. Add the samples to the RFS and reflash the NVIDIA DRIVE AGX Orin™ platform to make the new samples available there.

6.7.5.1.2 Running the NvMedia Samples

Prerequisites

- > The shared libraries, executables, and configuration files or scripts required to configure the cameras. For details, refer to the specific sample application.
- > The video files in case of decode/encode samples.

Deserializer Clock Control

Deserializer part on P3898 platform does not have dedicated oscillator. Clock input is received from the display serializer. Before launching the camera application, turn on RCLKOUT from the display serializer using the following commands,

```
echo 386 > /sys/class/gpio/export
echo out > /sys/class/gpio/PG.03/direction
cat /sys/class/gpio/PG.03/direction
echo 1 >/sys/class/gpio/PG.03/value
i2ccmd -d /dev/i2c-8 -a 0x40 -l 2 -o 0x03 -w 0x30
```

6.7.5.1.2.1 To run a sample application

If the /home/username/drive-linux/samples/nvmedia directory does not exist on the DRIVE AGX Orin Platform, and you wish to copy all samples, us the `copytarget-samples` command and reflash the board as described previously in [Building the NvMedia Samples](#).

If the /home/username/drive-linux/samples/nvmedia directory does not exist on the NVIDIA DRIVE AGX Orin™ platform, copy a single rebuilt sample:

1. Copy the sample to the target system. For example, from the target execute:

```
cd /home/<username>/
mkdir drive-linux/samples/nvmedia
rcp -r <host_user>@<host_ip>:<complete_host_install_path>
/drive-linux/samples/nvmedia/<sample_dir> /home/<username>/
drive-linux/samples/nvmedia
```

If the /home/username/drive-linux/samples/nvmedia directory already exists, update a single rebuilt sample using the previous procedure without the `mkdir` command.

2. Navigate to the sample application directory.
3. Copy the default configuration file and modify your copy for your needs.
4. Run the sample application.

6.7.5.2 SIPL Sample Applications

The SIPL sample applications demonstrate how to use the SIPL API to capture and process images from automotive cameras.

6.7.5.2.1 Tips for Using Sample Applications

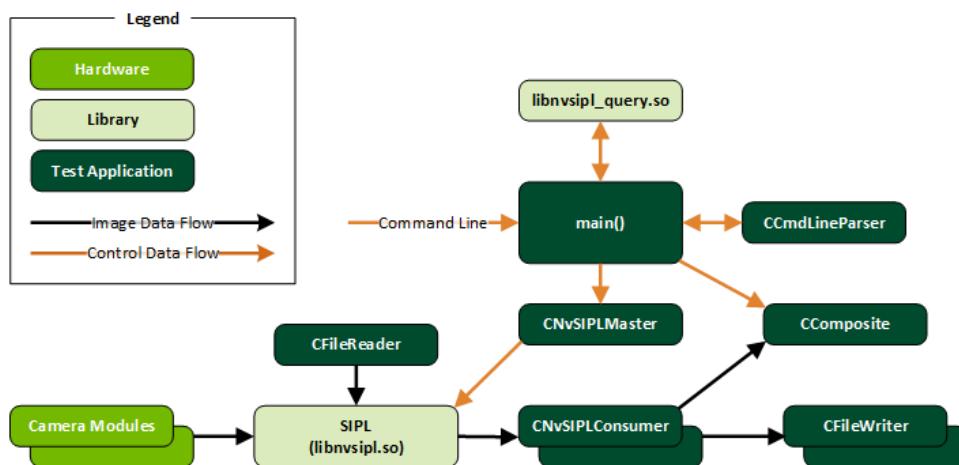
- When developing drivers for camera sensors, use the `nvsipl_camera` application as an example test application.
- Ensure that the Ethernet-based communication used for camera power control is configured correctly.

6.7.5.2.2 SIPL Camera (nvsipl_camera)

The NvMedia nvsipl_camera sample application demonstrates how to use the NvMedia SIPL API to select and initialize a camera platform configuration. The sample uses the NvSIPLQuery, NvSIPLDeviceBlock, NvSIPLCamera, and NvSIPLClient functions to program the image devices, NVIDIA® Tegra® VI and ISP hardware engines.

nvsipl_camera provides switches to select the platform configuration and customize the selected platform configuration by specifying the enable mask. For more information about the NvSIPL API, refer to the API documentation for [NvSIPL](#).

6.7.5.2.2.1 Architecture



The test application has a single `main()` function that coordinates all of the application's individual tasks. First, the command line switches are parsed using a `CCmdLineParser` object. Next, the platform configuration is obtained using the `INvSIPLQuery` interface and then an `INvSIPLCamera` instance is created and configured.

The `main()` thread creates `CNvSIPLConsumer` objects to receive the SIPL output buffers. Depending on the choice of the command line switches, the output images can be used in one of two ways: either written to a file or sent to a display. In the case of file writing, each `CNvSIPLConsumer` creates its own `CFileWriter` object to use for writing its received images to a file. For display, a `CComposite` object and its supporting classes (such as `CCompositeHelper`) are created to combine the images from a camera group to a single frame and then send that composite frame to the display.

Instead of receiving input from a camera module, the SIPL library can optionally read and process images from a file. When the appropriate command line switches are supplied, the application uses a `CFileReader` object to extract the images from a file and submit them to the SIPL library.

6.7.5.2.2.2 Running the application

Before you can run nvsipl_camera you must follow the procedures in [Building and Running the NvMedia Samples](#).

To run the sample on the target

1. Change the current directory to the folder that contains the nvsipl_camera binary:

```
samples/nvmedia/nvsipl/test/camera/
```

2. Enter one of these commands to launch the application:

> Without display:

```
$ ./nvsipl_camera -c "<platform>"
```

> With one display:

```
$ ./nvsipl_camera -c "<platform>" -d 1
```

> With two displays:

```
$ ./nvsipl_camera -c "<platform>" -d 2
```

Where <platform> is the platform configuration name.



Note:

Frame drops may occur when the connected display resolution is higher than 4K. This is not observed with 1080p displays.

To run the sample on the target as non-root

Change the current directory to the folder that contains the nvsipl_camera binary:

```
samples/nvmedia/nvsipl/test/camera/
```

Launch Command Construction

```
iolauncher --wait -U
1000:1000,10100,10140,10150,2110,2230,2340,2341,2342,2343,2347,2400,3000,3200,3220,3800,3810,3830
-A nonroot,allow,public_channel -A nonroot,allow,interruptevent -A
nonroot,allow,able=cdac/devblk:3 -A all,allow,able=nvsys/system_info -A
all,allow,able=nvcap/reserve_access_isp_channel -A all,allow,able=nvcap/
reserve_access_vi_channel -A all,allow,able=cba/tag:3 /samples/nvmedia/nvsipl_camera -c
V1SIM623S4RU5195NB3_CPHY_x4 -m "'0 1111 0 0'" --showfps
```

In the previous example, the primary UID:GID is 1000:1000, which is the default-installed non-privileged nvidia:nvidia user and group.

Required SGID for Camera Application

Following are the minimum set of SGIDs required for a SIPL Client:

SGID	Library
2110	nvsku
2230	nvcap
2340	nvi2c0
2341	nvi2c1
2342	nvi2c2

SGID	Library
2343	nvi2c3
2347	nvi2c7
2400	devv_nvhw
3000	nvsys
3220	io_pkt
3800	devm-cdac
10100	devg_nvrm_nvmap
10140	devg_nvrm_nvhost
10150	devg_nvrm_nvgpu_igpu
55006	libnvisp.so
55007	libnvisppg.so
55016	libnvscibuf.so.1
55017	libnvscicommon.so.1
55018	libnvscistream.so.1
55019	libnvscisync.so.1
55070	libnvsipl_devblk_campwr_max20087_fusa.so
55066	libnvsipl_devblk_drv_imx623vb2.so
55067	libnvsipl_devblk_drv_imx728vb2.so
55034	libnvsipl_devblk_drv_ar0820_cust1.so
40002	libc.so.5
40005	libm.so.3
40006	libslog2.so.1
45066	libnvdtcommon.so
40020	libudt.so
45047	libnvsciipc.so

SGID	Library
45046	libnvscievent.so
45031	libnvivc.so
45071	libnvsocsys.so
45112	libnvdvms_client.so
45038	libnvos.so
45037	libnvos_s3_safety.so
45069	libnvrm_mem.so
45043	libnvrm_host1x.so
45044	libnvrm_stream.so
45042	libnvrm_gpu.so
45045	libnvrm_surface.so
45085	NITO Files
40002	ldqnx-64.so.2, libgcc_s.so.1, libc.so.5
40052	libcatalog.so.1
40005	libm.so.3
40006	libslog2.so.1
45066	libnvdtcommon.so
40020	libudt.so
45047	libnvsciipc.so
45046	libnvscievent.so
45031	libnvivc.so
45071	libnvsocsys.so
45112	libnvdvms_client.so
45038	libnvos.so
45037	libnvos_s3_safety.so

SGID	Library
45069	libnvrm_mem.so
45043	libnvrm_host1x.so
45044	libnvrm_stream.so
45042	libnvrm_gpu.so
45045	libnvrm_surface.so
40052	libcatalog.so.1

Required QNX Abilities

Following are the minimum set of QNX abilities required for a SIPL client:

Custom Ability
public_channel
interruptevent
cdac/devblk:<i2c_bus>
nvsys/system_info
nvcap/reserve_access_isp_channel
nvcap/reserve_access_vi_channel
cba/tag:<i2c_bus>

<i2c_bus> corresponds to the i2c bus of deserializer. In the examples below the bus IDs are 0,2,3,7 for deserializer A, B, C and D respectively.

SIPL CDAC

SIPL CDAC Resource Block (virtualized deserializer) nodes are populated under /dev/sipl/resblockXY, the GID of the Resource Block nodes are configured the SIPL CDAC Device Tree for each Resource Block. Currently, the default parameters are set to the following for the Orin and similar platforms:

Resource Block	GID	Description
/dev/sipl/resblock0	3810	Deserializer A (i2c0)

Resource Block	GID	Description
/dev/sipl/resblock2	3830	Deserializer B (i2c2)
/dev/sipl/resblock3	3850	Deserializer C (i2c3)
/dev/sipl/resblock7	3870	Deserializer D (i2c7)

6.7.5.2.2.3 Display Layout

When nvsipl_camera runs with the -d option, the output images for one camera group display using the layout below. If an output is not enabled, the corresponding portion of the display is dark. You can use interactive commands to switch between the outputs of different camera groups.

If only one output is enabled on a camera group, it displays using the full screen.

 Note:	<ol style="list-style-type: none"> ISP Formats Usage: when -d option is used, all enabled ISP outputs are passed to VIC and composition occurs to obtain a RGB888 image that is pushed to display. VIC does not support composition for floating point or 16-bit images. The default ISP2 output format (RGBA PACKED FP16 PL as documented in the <i>NVIDIA DRIVE OS SDK API Reference Guide for INvSIPLCamera : :GetImageAttributes</i>, is overridden to be YUV 420 SEMI-PLANAR UINT8 BL when ISP2 output is requested. RGB-IR Sensor output: for RGB-IR sensors such as OV2312, ISP1 output cannot be displayed because the LUMA format is 16 bit and not supported by VIC. Display Interface: nvsipl_camera app uses OpenWFD display interfaces. This is not compatible with EGLDevice/X11/WinScreen/Wayland. VIC Bandwidth: when more than one output displays, while using 8MP camera, to preserve the limited VIC bandwidth, ISP downscales the images from 4k to 1080p before passing the images to VIC for format conversion.
--	--

RAW output Link 0	ISPO output Link 0	ISP1 output Link 0	ISP2 output Link 0
RAW output Link 1	ISPO output Link 1	ISP1 output Link 1	ISP2 output Link 1
RAW output Link 2	ISPO output Link 2	ISP1 output Link 2	ISP2 output Link 2
RAW output Link 3	ISPO output Link 3	ISP1 output Link 3	ISP2 output Link 3

6.7.5.2.2.4 Secondary Capture

In secondary capture mode, the secondary (processor B) captures images from the sensors that are already programmed by the primary (processor A).

When you run `nvsipl_camera` on processor B, you can use the `--enableSecondary` switch to capture in secondary mode.

Limitations

- > Before starting `nvsipl_camera` on the secondary, you must start capture on the primary. The secondary can continue to capture even after the primary has stopped capturing.
- > Enable masks specified using the `-m` command line switch must be identical for both primary and secondary. (See [Command Line Switches](#).)
- > Platform configuration specified using the `-c` command line switch must be identical for primary and secondary. (See [Command Line Switches](#).)

6.7.5.2.2.5 Command Line Switches

The following table shows the `nvsipl_camera` application's command line switches.

All numeric arguments may be specified in decimal (e.g., 18) or hexadecimal (e.g., `0x12`).

Switch	Description	Default Setting
<code>-h</code>	Displays help text.	Display only if an invalid command line argument is found.
<code>-c "name"</code>	Specifies name of platform configuration that describes the connection of image sensors to Orin-based platforms. Supported configurations are displayed by the <code>-h</code> switch.	Required switch.

Switch	Description	Default Setting
<code>--link-enable-masks "<m-AB> <m-CD> <m-EF> <m-GH>"</code>	<p>Enables masks for links on each deserializer connected to camera groups A, B, C, and D.</p> <p>The number of masks must equal the number of deserializers in the platform configuration selected with the <code>-c</code> switch.</p> <p>Each mask <code><m-></code> is a two-byte unsigned integer which applies to one deserializer and controls the four links that can be enabled for that deserializer. If the mask is expressed as a four-digit hexadecimal number, the last digit may be 1 to enable link 0 or 0 to disable it; the next-to-last may be 1 to enable link 1 or 0 to disable it; and so on.</p> <p>For example,</p> <p>"0x0000 0x1101 0x0000 0x0000" disables all links on the deserializers connected to camera groups A, C, and D, and enables links 0, 2, and 3 on the deserializer connected to camera group B.</p>	All links on all deserializers are enabled.
<code>-d <n></code>	Specifies the number of displays to send the output to. Supported values are 1 to 2.	Display output is disabled.
<code>-p "x0 y0 width height"</code>	Defines display position (upper left corner is at <i>x0, y0</i>) and dimensions (<i>width, height</i>) of the display rectangle.	Full screen display.

Switch	Description	Default Setting
<code>-f "prefix"</code>	<p>Sets a prefix for the output file's name. The name of the created file has the form:</p> <pre data-bbox="507 397 812 424"><p>_cam_<s>_out_<n>.<e></pre> <p>Where:</p> <ul style="list-style-type: none"> ➢ <code><p></code> is the prefix. It may begin with a pathname. ➢ <code><s></code> is the ID of the sensor that originated the images. ➢ <code><n></code> is the output number, which is 0 for RAW images, 1 for ISP0 output, and 2 for ISP1 output. ➢ <code><e></code> is the file extension, which is <code>raw</code> for RAW images and <code>yuv</code> for ISP output. <p>For example, if the prefix is <code>/home/nvidia/test</code>, the pathname of the RAW output file from Sensor0 is:</p> <pre data-bbox="507 973 943 1001">/home/nvidia/test_cam_0_out_0.raw</pre>	No output files are generated.
<code>-r <n></code>	Exits application after <i>n</i> seconds.	Application runs forever.
<code>-v <n></code>	Sets verbosity level. Supported values are 0 (only errors are printed) to 4 (maximum verbosity).	0
<code>-t "file"</code>	Specifies a custom platform configuration JSON file.	Application uses its built-in database of platform configurations.
<code>-l</code>	Lists platform configurations defined in the file specified by the <code>-t</code> switch.	
<code>-i "file"</code>	Enables simulator mode testing and specifies the RAW file to be used as input source for the SIPL library. <code>--link-enable-masks</code> must be used to enable only one camera module.	Simulator mode is disabled.
<code>--enableRawOutput</code>	Enables RAW output.	Disabled.

Switch	Description	Default Setting
--disableISP0Output	Disables ISP0 output.	Enabled.
--disableISP1Output	Disables ISP1 output.	Enabled.
--disableISP2Output	Disables ISP2 output.	Enabled.
--enablePassive	Enable passive mode.	Disabled
--showfps	Prints FPS (frames per second) messages on stdout.	Disabled.
--showmetadata	Shows metadata when RAW output is enabled.	Disabled.
--plugin <type>	Auto Control Plugin. Supported types (default: if NITO available 0, else 1): <ul style="list-style-type: none"> > 0: NVIDIA Auto Exposure/Auto White Balance (AE/AWB) Plugin > 1: Custom plug-in 0 	Defaults to 0 if the NITO file is available for all camera modules, else 1.
--autorecovery	Automatically attempts to recover any broken links on a periodic interval.	Disabled.
--nvsci	Uses NvSciStream for frame synchronization and transfer.	Disabled.
--profile "file-prefix"	Dumps profiling timestamps in <code>file-prefix_cam_x_out_y.csv</code> and shows frame rate (even if --showfps is not specified).	Disabled.
--skipFrames <val>	If using the -f switch, skips val frames at the beginning before writing to file.	No frames are skipped.
--writeFrames <val>	If using the -f switch, writes a maximum of val frames to file (starting after the number of skipped frames, if specified).	All captured frames are written to file.
--nito "folder"	Specifies the directory to search for NVIDIA Image Tuning Object (NITO) files.	Search folder is /usr/share/camera/.
--icrop "y+h"	Specifies the ISP input crop where input is cropped to bottom vertical offset y plus height h.	Disabled.

Switch	Description	Default Setting
--showEEPROM	Reads data (such as the sensor name and revision) from the camera module EEPROM, then displays that information on the command line.	Disabled.
--autoLED	Enables automatic LED control, specifically for AR0234.	Disabled.
--showNitoMetadata	Retrieves and displays NITO Metadata (ID, dataHash, schemaHash) for each parameter set(s) in the NITO file(s) loaded by the application to stdout.	Disabled.
--numParameterSetsInN <val>	Maximum number of parameter sets in NITO file. Must be greater than or equal to one (1). Use if expecting more than ten (10) parameter sets in NITO file(s).	10
--useNvSciBufPath	Enables use of NvSciBuf Buffers as inputs instead of NvMediaImages.	Disabled.
--ignoreError	Ignore the fatal error	Disabled.
--enableStatsOverrideTe	Enable ISP statistics settings override	Disabled.
--disableSubframe	Disable Subframe feature	Disabled
--enablePassive	Enable passive mode	Disabled.
--setSensorCharMode <expNo>	Set sensor in characterization mode with exposure number (only supported for AR0820)	Disabled.
--numParameterSetsInN <val>	Number of parameter sets in NITO file. Use if expecting >10 parameter sets in NITO file(s).	10

NVIDIA Image Tuning Object (NITO) is a binary file containing ISP settings, tuning, and characterization parameters for a specific camera module.

6.7.5.2.2.6 Interactive Menu Options

Option	Description
l d	Lists all camera groups in use by the application. Available only when command line switch -d is used to enable display.
e <g><d>	Displays the output of camera group g on display d. To see the available camera groups, use the l d option.
dl <sensor ID>	Disable link for sensor ID
el <sensor ID>	Enable link for sensor ID without module reset
elr <sensor ID>	Enable link for sensor ID with module reset
cm	Check the module availability
q	Quits the application.
les <sensor ID>	Enables the LED associated with the sensor ID for GazeT ARO234.
lds <sensor ID>	Disables the LED associated with the sensor ID for GazeT ARO234.
cust <sensor ID>,<value>	Sets an example value for the OV2311 custom API

6.7.5.2.2.7 Optional Features

Heterogenous Frame Synchronization



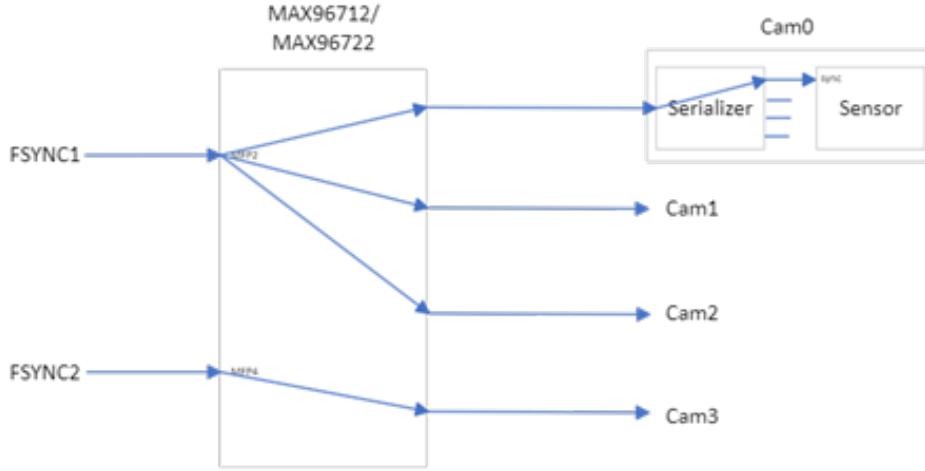
Note: This function requires hardware support, which is available on P3710-10-a04, p3710-10-s05, p3710-12-a04, and p3710-12-s05. Refer to [DRIVE Platform Supported Boards](#) for the board information.

TSC_EDGE_OUT signals are used as FSYNC signals supplied to sensors through the deserializer and the serializer, and 3 x TSC_EDGE_OUT signals are applicable on P3710-10-a04, p3710-10-s05, p3710-12-a04, p3710-12-s05. Each TSC_EDGE_OUT signal is preconfigured with the same or different frequencies in the device tree while the OS boots up, and they are supplied to each deserializer directly, or through the multipliers and multiplexers to provide the different FSYNC signal options for the different use cases.

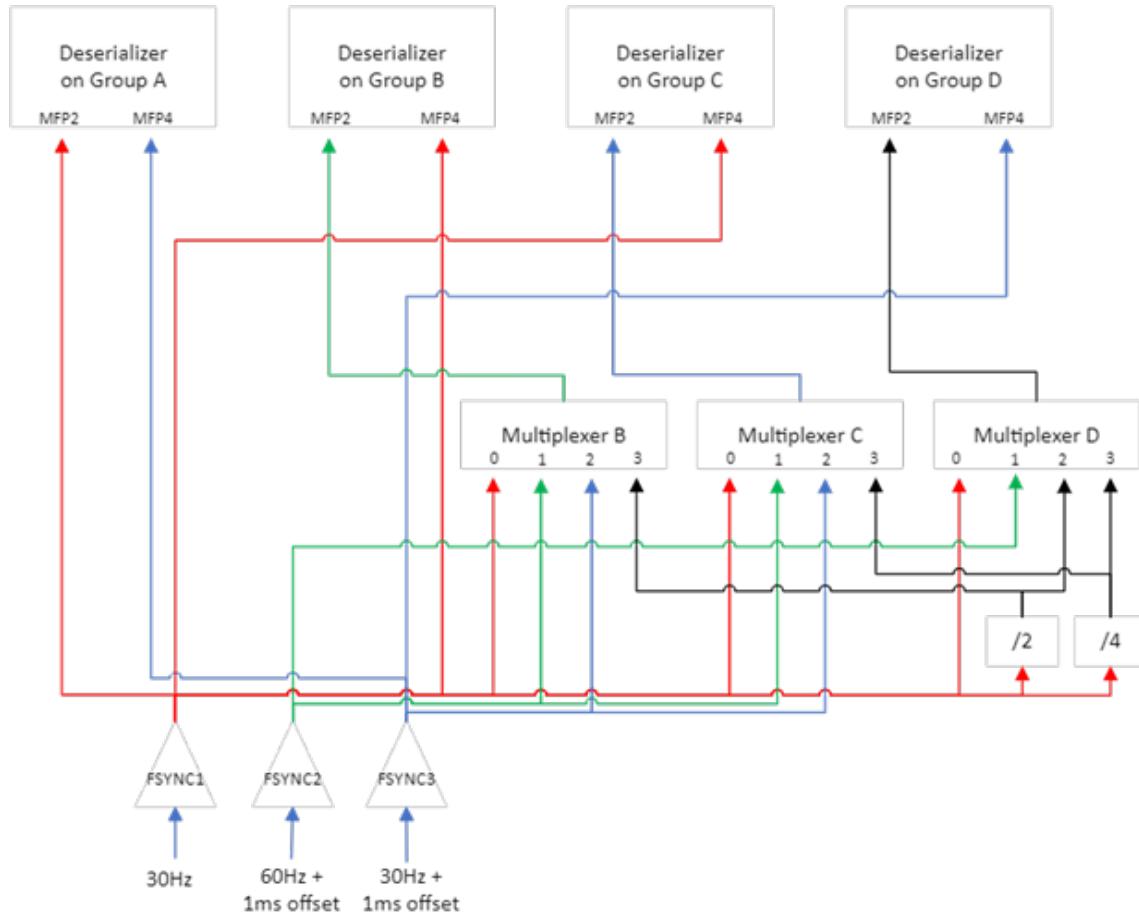
The deserializer selects one FSYNC and forwards it to the camera module selectively. The following image is an example use case to select the different FSYNC signal as a source of each camera module. The hardware block image, below, indicates how each FSYNC signal is routed. The timing diagram indicates what frequencies are configured and what offset is used among 3 x TSC_EDGE_OUT signals in the SDK and PDK by default. If

different frequencies or offsets are required, refer to [Fsync Signal Generation](#) for more information.

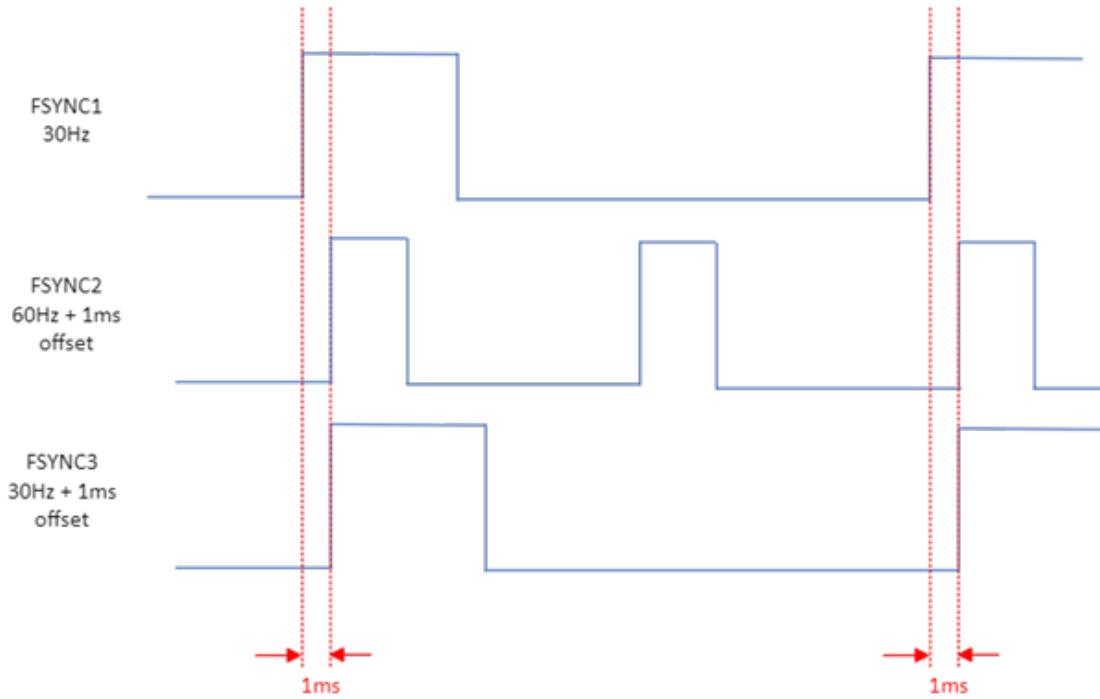
Example of forwarding two different FSYNC signals to different camera modules:



Hardware block diagram to support heterogenous frame synchronization:



Timing diagram among 3 x FSYNC signals:



Reference Codes

The deserializer driver (MAX96712DeserializerDriver_nv) provides a custom interface API, SetHeteroFrameSync, to select the MFP pins per link and to control the multiplexer, located in drive-<OS>/samples/nvmedia/nvsipl/devblk/devblk_new/devices/MAX96712DeserializerDriver_nv/CNvMMMax96712_Fusa_nv.cpp. The SetHeteroFrameSync API calls two sub APIs, MAX96712WriteParameters(), with CDI_WRITE_PARAM_CMD_MAX96712_SET_HETERO_FRAME_SYNC to select MFP of the deserializer as a FSYNC source for the camera module per link and DevBlkCDISetFsyncMux() to change the input source of the multiplexer associated with the specific deserializer.

The nvsipl_camera sample application (drive-<OS>/samples/nvmedia/nvsipl/test/camera) provides two command options to support the heterogeneous frame synchronization and calls the SetHeteroFrameSync API to configure the MFPs and the multiplexer per deserializer.

```
> --syncGPIOIdxDes 'gpios'
```

: MFP pin index is a list of MFP pins for each deserializer.

: Eg: '2 2 2 2 2 2 2 2 2 2 2 4 2 4' use MFP4 for the link 1 and 3 on CSI-GH interface and MFP2 is used for other links.

```
> --syncMuxSelDes 'Indexes'
```

: Selection index to the multiplexer. Applicable only for the specific board.

:Index is a list of selection index for each deserializer.

:Eg: '1 2 3' selects the selection index 1 for the multiplexer on the deserializer on the camera group B, the selection index 2 for the multiplexer on CSI-EF, and the selection index 3 for the multiplexer on CSI-GH.

Automatic Link Recovery

The automatic link recovery feature can be enabled by specifying the --autorecovery switch. If enabled, the nvsipl_camera application will keep track of any links that fail (for example, if a camera module is disconnected) and it will periodically attempt to recover that link to restore normal functionality. An example showing how to use this feature is provided below.

```
$ ./nvsipl_camera --platform-config "SF3324_DPHY_x4" --link-enable-masks "0x0001 0x0000
0x0000 0x0000" --showfps -d 1 --autorecovery
NvSci
```

NvSci utilities can be enabled by specifying the --nvsci switch. If enabled, the synchronization and transfer of frames from the NvSIP library to the display are managed using NvSciSync and NvSciStream. An example showing how to use this feature is provided below.

```
$ ./nvsipl_camera --platform-config "SF3324_DPHY_x4" --link-enable-masks "0x0001 0x0000
0x0000 0x0000" --showfps -d 1 --nvsci
```

Profiling

Performance metrics for nvsipl_camera can be displayed by specifying the --profile switch. The nvsipl_camera --profile command line option takes a file prefix as an argument and dumps the profiling timestamps into `file-prefix_cam_x_out_y.csv` for each camera and enabled output. The dumped CSV file has the following columns:

For Capture Output

Capture Timestamp (us)

For ISP Output

Capture Timestamp (us), Capture Done Event Timestamp (us)

Following is an example showing how to use this feature:

```
$ ./nvsipl_camera --platform-config "SF3324_DPHY_x4" --link-enable-masks "0x0001 0x0000
0x0000 0x0000" -d 0 --profile /tmp/sipl_profile
```

Fetch NITO Metadata

NITO Metadata can be fetched by specifying the --showNitoMetadata switch. If enabled, the NITO file loaded by the application will have its parameter set(s) ID, schema hash, and data hash retrieved and printed to stdout in the following format:

- > Parameter Set ID: "00000000-0000-0000-0000-000000000000" (in hex)
- > Schema/data hash: "0000..." (hex string with no spaces)

Note the API instantiates an array of structures to store the metadata retrieved for each parameter set in the NITO file, thus it requires that the length of this array is greater

than or equal to the number of the parameter sets in the NITO file. The app assumes by default a maximum of ten (10) parameter sets present in the NITO file.

If the user knows that greater than ten (10) parameter sets exist in the NITO file, then they can specify this using the `--numParameterSetsInNITO <val>` switch. Note that in order to specify this switch, the user must also specify the `--showNitoMetadata` switch.

An example showing how to use this feature, without setting a custom number of parameter sets, is provided below:

```
$ ./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4"
--link-enable-masks "0x0001 0x0000 0x0000 0x0000"
--showNitoMetadata
```

An example showing the error messages when the NITO file contains more parameter sets than the default of ten (10), is provided below:

```
$ ./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4"
--link-enable-masks "0x0001 0x0000 0x0000 0x0000"
--showNitoMetadata
```

Error messages if parameter sets retrieved from file is greater than ten (10):

```
... Call to fetchNitoMetadata API returned code:1,
valid API parameters not provided, check API arguments.
... GetNitoMetadataFromMemory: Call to fetchNitoMetadata API
failed, look at logs for error
... nvsipl_camera: ERROR: Failed to retrieve metadata from NITO file
... nvsipl_camera: ERROR: Failed to run Fetch NITO Metadata API
```

To resolve the above error, and in general if needed, an example showing how to use this feature when setting a custom number of parameter sets, is provided below:

```
$ ./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4"
--link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showNitoMetadata
--numParameterSetsInNITO 12
```

6.7.5.2.2.8 Examples

Platform configuration: V1SIM728S2RU4120HB20 modules in two-lane CPHY mode

This configuration assumes that 4 V1SIM728S2RU4120HB20 camera modules are connected to the camera group A and 4 V1SIM728S2RU4120HB20 camera modules are connected to the camera group D of the platform.

Example 2: Use link 0 of group A and link 2 of group D

```
$./nvsipl_camera -c "V1SIM728S2RU4120HB20_CPHY_x2" --link-enable-masks "0x0001 0x0000
0x0000 0x0100" -d 1
```

This command specifies V1SIM728S2RU4120HB20 modules on link 0 of the deserializer on camera group A and link 2 of the deserializer on camera group D. It displays output on one display.

Example 3: Use links 0 of group A and 2 of group D with secondary capture

1. On processor B, enter this command:

```
$./nvsipl_camera -c "V1SIM728S2RU4120HB20_CPHY_x2" --link-enable-masks "0x0001 0x0000
0x0000 0x0100" --enablePassive
```

This command specifies V1SIM728S2RU4120HB20 modules on link 0 of the deserializer on camera group A and link 2 of the deserializer on camera group D. It displays output on one display. nvsipl_camera application waits for the input from the user

2. On processor A, enter this command:

```
$./nvsipl_camera -c "V1SIM728S2RU4120HB20_CPHY_x2" --link-enable-masks "0x0001 0x0000
0x0000 0x0100" -d 1
```

This command specifies V1SIM728S2RU4120HB20 modules on link 0 of the deserializer on camera group A and link 2 of the deserializer on camera group D. It displays output on one display

3. Once nvsipl_camera application starts on the processor A, type any key on the process B to let nvsipl_camera application on process B starts to capture

Example: File input mode

```
$./nvsipl_camera -c "V1SIM728S2RU4120HB20_CPHY_x2" --link-enable-masks "0x0001 0x0000
0x0000 0x0000" -i <raw_file_name> -d 1
```

This command specifies an input file that contains RAW frames captured using V1SIM728S2RU4120HB20 modules. It displays output on one display.

6.7.5.2.3 Camera Commands

This section describes camera commands for supported cameras.

`nvsipl_camera` should run as root or with sudo.

6.7.5.2.3.1 Maxim Integrated GMSL SERDES

Note 1.

Two different deserializers are used per camera group, and each one supports the different link speed between the serializer and the deserializer. Connect the camera module to the compatible deserializer. Otherwise, the link lock error is reported.

P3710

Camera Group A : MAX96712(3 or 6Gbps)
 Camera Group B : MAX96722(3 Gbps)
 Camera Group C : MAX96712(3 or 6Gbps)
 Camera Group D : MAX96712(3 or 6Gbps)

P3663

Camera Group A : MAX96712(3 or 6Gbps)
 Camera Group B : MAX96722(3 Gbps)

P3898

Camera Group A : MAX96724(3 or 6Gbps)
 Camera Group B : MAX96724F(3 Gbps)

Note 2.

I2C bus speed is 1MHz for the camera group A and B on P3710 and P3663. Please make sure HW components in the camera module can support 1MHz I2C bus speed

AR0820 Using 1 Lane CPHY

Available module names:

- > Entron AR0820: F008A120RM0AV2

Command example:

```
./nvsipl_camera --platform-config "<module_name>_CPHY_x1" --link-enable-masks "0x0001
0x0000 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
A	1	<code>./nvsipl_camera --platform-config "F008A120RM0AV2_--link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1</code>	Y	Y
	2	<code>./nvsipl_camera --platform-config "F008A120RM0AV2_--link-enable-masks "0x0011 0x0000 0x0000 0x0000" --showfps -d 1</code>	Y	Y
C	1	<code>./nvsipl_camera --platform-config "F008A120RM0AV2_--link-enable-masks "0x0000 0x0000 0x0001 0x0000" --showfps -d 1</code>	Y	N

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
	2	./nvsipl_camera --platform-config "F008A120RM0AV2--link-enable-masks "0x0000 0x0000 0x0011 0x0000" --showfps -d 1	Y	N
D	1	./nvsipl_camera --platform-config "F008A120RM0AV2--link-enable-masks "0x0000 0x0000 0x0000 0x0001" --showfps -d 1	Y	N
	2	./nvsipl_camera --platform-config "F008A120RM0AV2--link-enable-masks "0x0000 0x0000 0x0000 0x0011" --showfps -d 1	Y	N

AR0820 Using 2 Lane CPHY

Available module names:

- > Entron AR0820: F008A120RM0AV2

Command example:

```
./nvsipl_camera --platform-config "<module_name>_CPHY_x2" --link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
A	1	<pre data-bbox="709 348 954 580"> ./nvsipl_camera --platform-config "F008A120RM0AV2" --link-enable- masks "0x0001 0x0000 0x0000 0x0000" -- showfps -d 1 </pre>	Y	Y
	2	<pre data-bbox="709 623 954 855"> ./nvsipl_camera --platform-config "F008A120RM0AV2" --link-enable- masks "0x0011 0x0000 0x0000 0x0000" -- showfps -d 1 </pre>	Y	Y
C	1	<pre data-bbox="709 918 954 1151"> ./nvsipl_camera --platform-config "F008A120RM0AV2" --link-enable- masks "0x0000 0x0000 0x0001 0x0000" -- showfps -d 1 </pre>	Y	N

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
	2	<pre data-bbox="714 318 959 608">./nvsipl_camera --platform-config "F008A120RM0AV2_--link-enable-masks "0x0000 0x0000 0x0011 0x0000" --showfps -d 1</pre>	Y	N
D	1	<pre data-bbox="714 608 959 897">./nvsipl_camera --platform-config "F008A120RM0AV2_--link-enable-masks "0x0000 0x0000 0x0000 0x0001" --showfps -d 1</pre>	Y	N
	2	<pre data-bbox="714 897 959 1182">./nvsipl_camera --platform-config "F008A120RM0AV2_--link-enable-masks "0x0000 0x0000 0x0000 0x0011" --showfps -d 1</pre>	Y	N

AR0820 Using 4 Lane CPHY

Available module names:

- > Entron AR0820: F008A120RM0A

Command example:

```
./nvsipl_camera --platform-config "<module_name>_CPHY_x4" --link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
A	1	<pre data-bbox="726 361 954 601"> ./nvsipl_camera -- platform-config "F008A120RM0A_CF" --link-enable- masks "0x0001 0x0000 0x0000 0x0000" -- showfps -d 1 </pre>	Y	Y
	4	<pre data-bbox="726 644 954 876"> ./nvsipl_camera -- platform-config "F008A120RM0A_CF" --link-enable- masks "0x1111 0x0000 0x0000 0x0000" -- showfps -d 1 </pre>	Y	Y
C	1	<pre data-bbox="726 939 954 1172"> ./nvsipl_camera -- platform-config "F008A120RM0A_CF" --link-enable- masks "0x0000 0x0000 0x0001 0x0000" -- showfps -d 1 </pre>	Y	N

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
	4	./nvsipl_camera --platform-config "F008A120RM0A_CFW" --link-enable-masks "0x0000 0x0000 0x1111 0x0000" --showfps -d 1	Y	N
D	1	./nvsipl_camera --platform-config "F008A120RM0A_CFW" --link-enable-masks "0x0000 0x0000 0x0001" --showfps -d 1	Y	N
	4	./nvsipl_camera --platform-config "F008A120RM0A_CFW" --link-enable-masks "0x0000 0x0000 0x0000 0x1111" --showfps -d 1	Y	N

IMX728 Using 2 Lane CPHY

Available module names:

P3663 and P3710

- > Valeo IMX728 B1 :
 - V728S1-120V1-FWC
 - V728S1-120V1-SCF
 - V728S1-070V1-SCR
 - V728S1-030V1-FTC
 - V728S1-030V1-RC
- > Valeo IMX728 B2.1P:
 - V1SIM728S1RU3120NB20
 - V1SIM728S1RU3070HB20
 - V1SIM728S1RU3070HB20_R
 - V1SIM728S1RU3030NB20
- > Valeo IMX728 B3.1C:

- V1SIM728S2RU1030NB30
- V1SIM728S2RU3070HB30
- V1SIM728S2RU3070HB30_R
- V1SIM728S2RU3120NB30
- V1SIM728S2RU3120HB30
- > Valeo IMX728 B2.2:
 - V1SIM728S2RU2030NB20
 - V1SIM728S2RU4070HB20
 - V1SIM728S2RU4070HB20_R
 - V1SIM728S2RU4120NB20
 - V1SIM728S2RU4120HB20
- > Valeo IMX728 CO:
 - V1SIM728S3RU4120NC00
 - V1SIM728S3RU4120HC00
 - V1SIM728S3RU4070HC00
 - V1SIM728S3RU4070HC00_R
 - V1SIM728S3RU2030NC00
- > Valeo IMX728 C1:
 - V1SIM728S3RU4120NC10
 - V1SIM728S3RU4120HC10
 - V1SIM728S3RU4070HC10
 - V1SIM728S3RU4070HC10_R
 - V1SIM728S3RU3030NC10
 - V1SIM728S3RU3030HC10
- > Valeo IMX728 C1 v2:
 - V1SIM728SARU4120NC10
 - V1SIM728SARU4120HC10
 - V1SIM728SARU4070HC10
 - V1SIM728SARU4070HC10_R
 - V1SIM728SARU3030NC10
 - V1SIM728SARU3030HC10
- > Smartlead IMX728 B1:
 - ROSIM728S2RU2120NB1
 - ROSIM728S2RU1070NB1
 - ROSIM728S2RU2030NB1
- > Smartlead IMX728 B2:
 - ROSIM728S3RU2120NB2
 - ROSIM728S3RU1070NB2
 - ROSIM728S3RU2030NB2
- > Continental IMX728 B2.1:

- C2SIM728S2RU1120NB20
- C2SIM728S2RU1070HB20
- C2SIM728S2RU1070NB20
- C2SIM728S2RU1030NB20

Command example:

```
./nvsipl_camera --platform-config "<module_name>_CPHY_x2" --link-enable-masks "0x0001
0x0000 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin (P3663)
A	1	./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x2" --link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1	Y	Y
	2	./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x2" --link-enable-masks "0x0011 0x0000 0x0000 0x0000" --showfps -d 1	Y	Y
C	1	./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x2" --link-enable-masks "0x0000 0x0000 0x0001 0x0000" --showfps -d 1	Y	N/A

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin (P3663)
	2	<code>./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x2" --link-enable-masks "0x0000 0x0000 0x0011 0x0000" --showfps -d 1</code>	Y	N/A
D	1	<code>./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x2" --link-enable-masks "0x0000 0x0000 0x0000 0x0001" --showfps -d 1</code>	Y	N/A
	2	<code>./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x2" --link-enable-masks "0x0000 0x0000 0x0000 0x0011" --showfps -d 1</code>	Y	N/A

P3898

- > Valeo IMX728 B2.1P:
 - V1SIM728S1RU3120NB20
 - V1SIM728S1RU3070HB20
 - V1SIM728S1RU3030NB20
- > Valeo IMX728 B3.1C:
 - V1SIM728S2RU1030NB30
 - V1SIM728S2RU3070HB30
 - V1SIM728S2RU3120NB30
 - V1SIM728S2RU3120HB30
- > Valeo IMX728 B2.2:
 - V1SIM728S2RU2030NB20
 - V1SIM728S2RU4070HB20
 - V1SIM728S2RU4120NB20
 - V1SIM728S2RU4120HB20

Command example:

```
./nvsipl_camera --platform-config "<module_name>_MAX96724_CPHY_x2" --link-enable-masks "0x0001 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin-N (P3898)
A	1	<code>./nvsipl_camera -c V1SIM728S2RU1030NB30 --link-enable-masks '0x1' --showfps -v 2</code>	Y
A	2	<code>./nvsipl_camera --platform-config "V1SIM728S2RU1030NB30" --link-enable-masks "0x0000 0x0011" --showfps -d 1</code>	Y



Note: nvsipl_camera application looks for PTP interface clock base for profiling. To enable this networking interface on P3898, refer to the P3898 Networking section.



Note: cameras with _R suffix have a rotated output of 180 degrees.

IMX728 Using 4 Lane CPHY

Available module names:

P3663 and P3710

- > Valeo IMX728 B1:
 - V728S1-120V1-FWC
 - V728S1-120V1-SCF
 - V728S1-070V1-SCR
 - V728S1-030V1-FTC
 - V728S1-030V1-RC
- > Valeo IMX728 B2.1P:
 - V1SIM728S1RU3120NB20
 - V1SIM728S1RU3070HB20
 - V1SIM728S1RU3070HB20_R
 - V1SIM728S1RU3030NB20
- > Valeo IMX728 B3.1C:
 - V1SIM728S2RU1030NB30
 - V1SIM728S2RU3070HB30
 - V1SIM728S2RU3070HB30_R
 - V1SIM728S2RU3120NB30

- V1SIM728S2RU3120HB30
- > Valeo IMX728 B2.2:
 - V1SIM728S2RU2030NB20
 - V1SIM728S2RU4070HB20
 - V1SIM728S2RU4070HB20_R
 - V1SIM728S2RU4120NB20
 - V1SIM728S2RU4120HB20
- > Valeo IMX728 C0:
 - V1SIM728S3RU4120NC00
 - V1SIM728S3RU4120HC00
 - V1SIM728S3RU4070HC00
 - V1SIM728S3RU4070HC00_R
 - V1SIM728S3RU2030NC00
- > Valeo IMX728 C1:
 - V1SIM728S3RU4120NC10
 - V1SIM728S3RU4120HC10
 - V1SIM728S3RU4070HC10
 - V1SIM728S3RU4070HC10_R
 - V1SIM728S3RU3030NC10
 - V1SIM728S3RU3030HC10
- > Valeo IMX728 C1 v2:
 - V1SIM728SARU4120NC10
 - V1SIM728SARU4120HC10
 - V1SIM728SARU4070HC10
 - V1SIM728SARU4070HC10_R
 - V1SIM728SARU3030NC10
 - V1SIM728SARU3030HC10
- > Smartlead IMX728 B1:
 - ROSIM728S2RU2120NB1
 - ROSIM728S2RU1070NB1
 - ROSIM728S2RU2030NB1
- > Smartlead IMX728 B2:
 - ROSIM728S3RU2120NB2
 - ROSIM728S3RU1070NB2
- > Continental IMX728 B2.1:
 - C2SIM728S2RU1120NB20
 - C2SIM728S2RU1070HB20
 - C2SIM728S2RU1070NB20
 - C2SIM728S2RU1030NB20

Command example:

```
./nvsipl_camera --platform-config "<module_name>_CPHY_x4" --link-enable-masks "0x0001
0x0000 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
A	1	<code>./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4" --link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1</code>	Y	Y
	4	<code>./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4" --link-enable-masks "0x1111 0x0000 0x0000 0x0000" --showfps -d 1</code>	Y	Y
C	1	<code>./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4" --link-enable-masks "0x0000 0x0000 0x0001 0x0000" --showfps -d 1</code>	Y	N/A

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
	4	<pre data-bbox="714 354 938 587"> ./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4" --link-enable-masks "0x0000 0x0000 0x1111 0x0000" --showfps -d 1 </pre>	Y	N/A
D	1	<pre data-bbox="714 644 938 876"> ./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4" --link-enable-masks "0x0000 0x0000 0x0000 0x0001" --showfps -d 1 </pre>	Y	N/A
	4	<pre data-bbox="714 933 938 1165"> ./nvsipl_camera --platform-config "V728S1-120V1-FWC_CPHY_x4" --link-enable-masks "0x0000 0x0000 0x0000 0x1111" --showfps -d 1 </pre>	Y	N/A

P3898

- > Valeo IMX728 B2.1P:
 - V1SIM728S1RU3120NB20
 - V1SIM728S1RU3070HB20
 - V1SIM728S1RU3030NB20
- > Valeo IMX728 B3.1C:
 - V1SIM728S2RU1030NB30
 - V1SIM728S2RU3070HB30
 - V1SIM728S2RU3120NB30
 - V1SIM728S2RU3120HB30
- > Valeo IMX728 B2.2:
 - V1SIM728S2RU2030NB20
 - V1SIM728S2RU4070HB20
 - V1SIM728S2RU4120NB20
 - V1SIM728S2RU4120HB20

Command example:

```
./nvsipl_camera --platform-config "<module_name>_MAX96724_CPHY_x4" --link-enable-masks "0x0001 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin-N (P3898)
A	1	<code>./nvsipl_camera -c V1SIM728S2RU1030NB30 --link-enable-masks '0x0001 0x0000' --showfps -v 2</code>	Y
A	2	<code>./nvsipl_camera --platform-config "V1SIM728S2RU1030NB30" --link-enable-masks "0x0011 0x0000" --showfps -d 1</code>	Y



Note: nvsipl_camera application looks for PTP interface clock base for profiling. To enable this networking interface on P3898, refer to the P3898 Networking section.



Note: cameras with _R suffix have a rotated output of 180 degrees

IMX623 Using 2 Lane CPHY

Available module names:

P3663 and P3710

- > Valeo IMX623 B1: V623S2-195V1-SVS
- > Valeo IMX623 B2:
 - V1SIM623S3RU3200NB20
 - V1SIM623S3RU3200NB20_R
- > Valeo IMX623 B3.2:
 - V1SIM623S4RU5195NB3
 - V1SIM623S4RU5195NB3_R
- > Valeo IMX623 C0:
 - V1SIM623S5RU5195NC0
 - V1SIM623S5RU5195NC0_R
- > Valeo IMX623 C1:
 - V1SIM623S5RU6195NC1
 - V1SIM623S5RU6195NC1_R
- > Valeo IMX623 C1 v2:
 - V1SIM623SARU6195NC1
 - V1SIM623SARU6195NC1_R

- > Smartlead IMX623 B1: ROSIM623S3RU1197NB1
- > Smartlead IMX623 B2: ROSIM623S4RU1197NB2
- > Continental IMX623 B2.1: C2SIM623S2RU1195NB20

Command example:

```
./nvspipl_camera --platform-config "<module_name>_CPHY_x2" --link-enable-masks "0x0000
0x0001 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
B	1	<code>./nvspipl_camera --platform-config "V623S2-195V1-SVS_CPHY_x2" --link-enable-masks "0x0000 0x0001 0x0000 0x0000" --showfps -d 1</code>	Y	Y
	2	<code>./nvspipl_camera --platform-config "V623S2-195V1-SVS_CPHY_x2" --link-enable-masks "0x0000 0x0011 0x0000 0x0000" --showfps -d 1</code>	Y	Y

P3898

- > Valeo IMX623 B3.2: V1SIM623S4RU5195NB3
- > Valeo IMX623 B2: V1SIM623S3RU3200NB20

Command example:

```
./nvsipl_camera --platform-config "<module_name>_MAX96724_CPHY_x2" --link-enable-masks "0x0000 0x0001" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin-N (P3898)
B	1	<code>./nvsipl_camera -c V1SIM623S3RU3200NB20 --link-enable-masks '0x0000 0x0001' --showfps -v 2</code>	Y
B	4	<code>./nvsipl_camera --platform-config "V1SIM623S3RU3200NB20 --link-enable-masks "0x0000 0x1111" --showfps -d 1</code>	Y



Note: nvsipl_camera application looks for PTP interface clock base for profiling. To enable this networking interface on P3898 refer to the P3898 Networking section.



Note: cameras with _R suffix have a rotated output of 180 degrees.

IMX623 Using 4 Lane CPHY

Available module names:

P3663 and P3710

- > Valeo IMX623 B1: V623S2-195V1-SVS
- > Valeo IMX623 B2:
 - V1SIM623S3RU3200NB20
 - V1SIM623S3RU3200NB20_R
- > Valeo IMX623 B3.2:
 - V1SIM623S4RU5195NB3
 - V1SIM623S4RU5195NB3_R
- > Valeo IMX623 C0:
 - V1SIM623S5RU5195NC0
 - V1SIM623S5RU5195NC0_R
- > Valeo IMX623 C1:
 - V1SIM623S5RU6195NC1
 - V1SIM623S5RU6195NC1_R
- > Valeo IMX623 C1 v2:
 - V1SIM623SARU6195NC1
 - V1SIM623SARU6195NC1_R
- > Smartlead IMX623 B1 : ROSIM623S3RU1197NB1

- > Smartlead IMX623 B2: ROSIM623S4RU1197NB2
- > Continental IMX623 B2.1: C2SIM623S2RU1195NB20

Command example:

```
./nvsipl_camera --platform-config "<module_name>_CPHY_x4" --link-enable-masks "0x0000
0x0001 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
B	1	./nvsipl_camera --platform-config "V623S2-195V1-SVS_CPHY_x4" --link-enable-masks "0x0000 0x0001 0x0000 0x0000" --showfps -d 1	Y	Y
	4	./nvsipl_camera --platform-config "V623S2-195V1-SVS_CPHY_x4" --link-enable-masks "0x0000 0x1111 0x0000 0x0000" --showfps -d 1	Y	Y

P3898

- > Valeo IMX623 B3.2: V1SIM623S4RU5195NB3
- > Valeo IMX623 B2: V1SIM623S3RU3200NB20

Command example:

```
./nvsipl_camera --platform-config "<module_name>_MAX96724_CPHY_x4" --link-enable-masks "0x0000 0x0001" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin-N (P3898)
B	1	<code>./nvsipl_camera -c V1SIM623S3RU3200NB20 --link-enable-masks '0x1' --showfps -v 2</code>	Y
B	4	<code>./nvsipl_camera --platform-config "V1SIM623S3RU3200NB20" --link-enable-masks "0x0000 0x1111" --showfps -d 1</code>	Y



Note: nvsipl_camera application looks for PTP interface clock base for profiling. To enable this networking interface on P3898, refer to the P3898 Networking section.



Note: cameras with _R suffix have a rotated output of 180 degrees.

OV2311 Using 2 Lane DPHY/CPHY

Available module names:

- > Leopard OV2311 A1: (DPHY only)
 - LI-OV2311-VCSEL-GMSL2-60H
 - LI-OV2311-VCSEL-GMSL2-60H_L
- > Leopard OV2311 B1: (CPHY only)
 - I00023111CML1050NB10
 - I00023111CML1050NB10_30FPS
- > Leopard OV2311 B2: (CPHY only)
 - LI-OV2311-VCSEL-GMSL2-55H
 - LI-OV2311-VCSEL-GMSL2-55H_L

Command example:

```
./nvsipl_camera --platform-config "<module_name>_DPHY_x2" --link-enable-masks "0x0000 0x0000 0x0001" --showfps -d 1
or
./nvsipl_camera --platform-config "<module_name>_CPHY_x2" --link-enable-masks "0x0000 0x0000 0x0001" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
C	1	<code>./nvsipl_camera --platform-config "LI-OV2311-VCSEL-GMSL2-60H_DPHY_x2" --link-enable-masks "0x0000 0x0000 0x0001 0x0000" --showfps -d 1 --enableRawOutput --disableISP0Output --disableISP1Output</code>	Y	N
	4	<code>./nvsipl_camera --platform-config "LI-OV2311-VCSEL-GMSL2-60H_DPHY_x2" --link-enable-masks "0x0000 0x0000 0x1111 0x0000" --showfps -d 1 --enableRawOutput --disableISP0Output --disableISP1Output</code>	Y	N
D	1	<code>./nvsipl_camera --platform-config "LI-OV2311-VCSEL-GMSL2-60H_DPHY_x2" --link-enable-masks "0x0000 0x0000 0x0000 0x0001" --showfps -d 1 --enableRawOutput --disableISP0Output --disableISP1Output</code>	Y	N
	4	<code>./nvsipl_camera --platform-config "LI-OV2311-VCSEL-GMSL2-60H_DPHY_x2" --link-enable-masks "0x0000 0x0000 0x0000 0x1111" --showfps -d 1 --enableRawOutput --disableISP0Output --disableISP1Output</code>	Y	N

OV2311 Using 4 Lane DPHY/CPHY

Available module names:

- > Leopard OV2311 A1: (DPHY only)
 - LI-OV2311-VCSEL-GMSL2-60H
 - LI-OV2311-VCSEL-GMSL2-60H_L
- > Leopard OV2311 B1: (CPHY only)
 - I00023111CML1050NB10
 - I00023111CML1050NB10_30FPS
- > Leopard OV2311 B2: (CPHY only)
 - LI-OV2311-VCSEL-GMSL2-55H
 - LI-OV2311-VCSEL-GMSL2-55H_L

Command example:

```
./nvsipl_camera --platform-config "<module_name>_DPHY_x2" --link-enable-masks "0x0000 0x0000 0x0000 0x0001" --showfps -d 1
or
./nvsipl_camera --platform-config "<module_name>_CPHY_x2" --link-enable-masks "0x0000 0x0000 0x0000 0x0001" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
C	1	<pre data-bbox="714 318 959 796"> ./nvspipl_camera --platform-config "LI-OV2311-VCSEL-GMSL2-60H_DPHY" --link-enable-masks "0x0000 0x0000 0x0001 0x0000" -- showfps -d 1 --enableRawOutput -- disableISP0Output -- disableISP1Output </pre>	Y	N

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
	4	<pre>./nvspil_camera --platform-config "LI-OV2311-VCSEL-GMSL2-60H_DPHY_" --link-enable-masks "0x0000 0x0000 0x1111 0x0000" -- showfps -d 1 -- enableRawOutput -- disableISP0Output -- disableISP1Output</pre>	Y	N
D	1	<pre>./nvspil_camera --platform-config "LI-OV2311-VCSEL-GMSL2-60H_DPHY_" --link-enable-masks "0x0000 0x0000 0x0001 0x0000" -- showfps -d 1 -- enableRawOutput -- disableISP0Output -- disableISP1Output</pre>	Y	N
	4	<pre>./nvspil_camera --platform-config "LI-OV2311-VCSEL-GMSL2-60H_DPHY_" --link-enable-masks "0x0000 0x0000 0x1111 0x0000" -- showfps -d 1 -- enableRawOutput -- disableISP0Output -- disableISP1Output</pre>	Y	N

OX5B Using 4 Lane CPHY

Available module names:

> Smartlead OX5B B2: R00OX05B1CHU1170NB20

Command example:

```
./nvsipl_camera --platform-config "<module_name>_CPHY_x4" --link-enable-masks "0x0001
0x0000 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
A	1	./nvsipl_camera --platform-config "R00OX05B1CHU11" --link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1	Y	Y
	4	./nvsipl_camera --platform-config "R00OX05B1CHU11" --link-enable-masks "0x1111 0x0000 0x0000 0x0000" --showfps -d 1	Y	Y
C	1	./nvsipl_camera --platform-config "R00OX05B1CHU11" --link-enable-masks "0x0000 0x0000 0x0001 0x0000" --showfps -d 1	Y	N/A

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
	4	<pre data-bbox="714 354 953 587"> ./nvsipl_camera --platform-config "R00OX05B1CHU11 --link-enable- masks "0x0000 0x0000 0x1111 0x0000" -- showfps -d 1 </pre>	Y	N/A
D	1	<pre data-bbox="714 644 953 876"> ./nvsipl_camera --platform-config "R00OX05B1CHU11 --link-enable- masks "0x0000 0x0000 0x0000 0x0001" -- showfps -d 1 </pre>	Y	N/A
	4	<pre data-bbox="714 914 953 1146"> ./nvsipl_camera --platform-config "R00OX05B1CHU11 --link-enable- masks "0x0000 0x0000 0x0000 0x1111" -- showfps -d 1 </pre>	Y	N/A

6.7.5.2.3.2 TI FPD-LINK SERDES

Note 1.

Connect the camera module to the correct deserializer, otherwise, the link lock error is reported.

P3710

Camera Group A: DS90UB9724 (7.55 Gbps)

Camera Group B: DS90UB9724 (7.55 Gbps)

Camera Group C: DS90UB9724 (7.55 Gbps)

Camera Group D: DS90UB9724 (7.55 Gbps)

Note 2.

I2C bus speed is 1MHz for the camera group A and B on P3710 and P3663. Please make sure HW components in the camera module can support 1MHz I2C bus speed

IMX728 FPD-Link Using 4 Lane CPHY

Available module names:

- IMX728 FPD-Link:

Command example:

```
./nvsipl_camera --platform-config "IMX728_FPDLINK_RGGB_CPHY_x4" --link-enable-masks
"0x0001 0x0000 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
A	1	<pre>./nvsipl_camera --platform-config "IMX728_FPDLINK_F --link-enable- masks "0x0001 0x0000 0x0000 0x0000" -- showfps -d 1</pre>	Y	N
	4	<pre>./nvsipl_camera --platform-config "IMX728_FPDLINK_F --link-enable- masks "0x1111 0x0000 0x0000 0x0000" -- showfps -d 1</pre>	Y	N
B	1	<pre>./nvsipl_camera --platform-config "IMX728_FPDLINK_F --link-enable- masks "0x0000 0x0001 0x0000 0x0000" -- showfps -d 1</pre>	Y	N
	4	<pre>./nvsipl_camera --platform-config "IMX728_FPDLINK_F --link-enable- masks "0x0000 0x1111 0x0000 0x0000" -- showfps -d 1</pre>	Y	N
C	1	<pre>./nvsipl_camera --platform-config "IMX728_FPDLINK_F --link-enable- masks "0x0000 0x0000 0x0001 0x0000" -- showfps -d 1</pre>	Y	N

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
	4	<pre data-bbox="714 318 964 608">./nvsipl_camera --platform-config "IMX728_FPDLINK_F" --link-enable-masks "0x0000 0x0000 0x1111 0x0000" --showfps -d 1</pre>	Y	N
D	1	<pre data-bbox="714 608 964 897">./nvsipl_camera --platform-config "IMX728_FPDLINK_F" --link-enable-masks "0x0000 0x0000 0x0001" --showfps -d 1</pre>	Y	N
	4	<pre data-bbox="714 897 964 1180">./nvsipl_camera --platform-config "IMX728_FPDLINK_F" --link-enable-masks "0x0000 0x0000 0x0000 0x1111" --showfps -d 1</pre>	Y	N

DS90UB971 SER TPG Using 4 Lane CPHY

Available module names:

- > DS90UB971 TPG:
 - DS90UB971_TPG_SENSOR

Command example:

```
./nvsipl_camera --platform-config "DS90UB971_RAW12_TPG_CPHY_x4" --link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
A	1	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1</pre>	Y	N
	4	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x1111 0x0000 0x0000 0x0000" --showfps -d 1</pre>	Y	N
B	1	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0000 0x0001 0x0000 0x0000" --showfps -d 1</pre>	Y	N
	4	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0000 0x1111 0x0000 0x0000" --showfps -d 1</pre>	Y	N
C	1	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0000 0x0000 0x0001 0x0000" --showfps -d 1</pre>	Y	N

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
	4	./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0000 0x0000 0x1111 0x0000" --showfps -d 1	Y	N
D	1	./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0000 0x0000 0x0000 0x0001" --showfps -d 1	Y	N
	4	./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0000 0x0000 0x0000 0x1111" --showfps -d 1	Y	N

DS90UB971 SER TPG Using 4 Lane DPHY

Available module names:

- > DS90UB971 TPG:
 - DS90UB971_TPG_SENSOR

Command example:

```
./nvsipl_camera --platform-config "DS90UB971_RAW12_TPG_DPHY_x4" --link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1
```

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
A	1	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0001 0x0000 0x0000 0x0000" --showfps -d 1</pre>	Y	N
	4	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x1111 0x0000 0x0000 0x0000" --showfps -d 1</pre>	Y	N
B	1	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0000 0x0001 0x0000 0x0000" --showfps -d 1</pre>	Y	N
	4	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0000 0x1111 0x0000 0x0000" --showfps -d 1</pre>	Y	N
C	1	<pre>./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable-masks "0x0000 0x0000 0x0001 0x0000" --showfps -d 1</pre>	Y	N

Camera Group	No. of Cams	Command	NVIDIA DRIVE AGX Orin™ Dev Kit (P3710-10)	NVIDIA DRIVE AGX Orin™ (P3663)
	4	<pre data-bbox="714 354 959 587"> ./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable- masks "0x0000 0x0000 0x1111 0x0000" -- showfps -d 1 </pre>	Y	N
D	1	<pre data-bbox="714 644 959 876"> ./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable- masks "0x0000 0x0000 0x0000 0x0001" -- showfps -d 1 </pre>	Y	N
	4	<pre data-bbox="714 912 959 1144"> ./nvsipl_camera --platform-config "DS90UB971_RAW1 --link-enable- masks "0x0000 0x0000 0x0000 0x1111" -- showfps -d 1 </pre>	Y	N

6.7.5.2.4 SIPL Reprocess (nvsipl_reprocess)

The NvMedia nvsipl_reprocess is a sample application that:

- Reads RAW images from the file system.
- Processes the RAW images through the Image Signal Processor (ISP).
- Writes the processed images to the file system.

6.7.5.2.4.1 Running the Application

1. Change to the folder that contains the nvsipl_reprocess binary:

```
samples/nvmedia/nvsipl/test/reprocess/
```

2. Enter the following command to launch the application:

```
$ ./nvsipl_reprocess -c "<platform>"
--link-enable-masks "<mask>" --out0 <format> -i <input> --nito <file>
```

Where:

- <platform> is the platform configuration name.
- <mask> specifies a sensor from a given camera group.
- <format> is the ISP output format.

- > <input> specifies the path of the RAW file to process.
- > <file> specifies the NVIDIA ISP Tuning Object to use.

The parameters above are required to run `nvsipl_reprocess`. The NVIDIA ISP Tuning Object (NITO) is a binary file containing ISP settings, tuning, and characterization parameters for a specific camera module. The NITO files are located at:

```
/usr/share/camera/
```

Sensor Module	Corresponding NITO file
V728S1-120V1-FWC_CPHY_x2	V728S1-120V1-FWC.nito
V728S1-120V1-FWC_CPHY_x4	V728S1-120V1-FWC.nito

6.7.5.2.4.2 Command Line Switches

The following table shows the `nvsipl_reprocess` application's command line switches.

All numeric arguments may be specified in decimal (for example, 18) or hexadecimal (for example, 0x12).

Switch	Description	Default Setting
-h or --help	Displays help text.	Display only if an invalid command line argument is found.
-c or --platform-config <name>	Specifies name of the platform configuration that describes the connection of image sensors to Xavier based platforms. The -h switch displays supported configurations.	Required argument.
-t or --test-config-file <file>	Set custom platform config JSON file name.	Use default platform config.
-v or --verbosity <n>	Sets verbosity level. Supported values are 0 (none) to 4 (maximum verbosity).	1 (errors)

Switch	Description	Default Setting
<code>-f or --filedump-prefix "<prefix>"</code>	<p>Sets a prefix for the output files' filenames.</p> <p>The output files contain the processed images.</p> <p>The names of the files created have the form:</p> <p style="padding-left: 40px;"><code><e>_<p>_cam_<s>.yuv</code></p> <p>Where:</p> <ul style="list-style-type: none"> > <code><e></code> is the prefix. It may begin with a pathname. > <code><p></code> contains the ISP output index. > <code><s></code> is the ID of the sensor that originated the images. <p>For example, if the prefix is /home/nvidia/test, the path name of the output file from Sensor0 processed by ISP1 is:</p> <p style="padding-left: 40px;"><code>/home/nvidia/test_ISP1_cam_0.yuv</code></p>	No output files are generated.
<code>-s or --frames-to-skip <s></code>	Skip <code><s></code> frames from the beginning of the file.	No frame skipping.
<code>-n or --frames-to-save <n></code>	Saves <code><n></code> frames after skipped frames.	Processing saves frames until the end of the input file.
<code>-i or --input-raw-file <file></code>	Specifies the RAW input file to process.	Required argument.

Switch	Description	Default Setting
<p>-m or --link-enable-masks <code><m-AB> <m-CD> <m-EF> <m-GH></code></p>	<p>Specifies the position of sensor at the time of raw capture. The application retrieves the configuration of the sensor using the mask.</p> <p>This application only supports one link at a time.</p> <p>For example, '0x0000 0x0001 0x0000 0x0000' specifies link 0 connected to the CSI-CD interface.</p>	Required argument.
<p>--icrop <code><y+h></code></p>	<p>Specifies the cropping at the input in the format <code><y+h></code>.</p>	
<p>-l or --list-configs</p>	<p>List config from file specified by -t or --test-config-file</p>	
<p>--out0 <code><format></code></p>	<p>Specifies the output format for processed images through ISP0.</p> <p>Supported formats: <code>yuv420_8b</code> (BL), <code>yuv420_8b_pl</code>, <code>yuv420_16b_709er</code> (BL), <code>yuv420_16b_709er_pl</code>, <code>yuv444_8b</code>, <code>yuv444_16b_709er</code>, <code>yuv444_8b_bl</code>.</p> <p>If both out0 and out1 are enabled, then their formats should match.</p>	<p>At least one of the three --out0 <code><format></code> or --out1 <code><format></code> or --out2 <code><format></code> must be present.</p>
<p>--out1 <code><format></code></p>	<p>Specifies the output format for processed images through ISP1.</p> <p>Supported formats: supported formats same as out0 for RGB Bayer sensor data</p> <p><code>luma_16b</code> in case of RGB-IR sensor data.</p> <p>If both out0 and out1 are enabled, then their formats should match if the sensor data input is from RGB Bayer sensor.</p>	<p>At least one of the three --out0 <code><format></code> or --out1 <code><format></code> or --out2 <code><format></code> must be present.</p>

Switch	Description	Default Setting
--out2 <format>	<p>Specifies the output format for processed images through ISP2.</p> <p>Supported formats: supported formats same as out0 and additionally rgb_sensor_16f for RGB Bayer sensor data.</p> <p>Not supported for RGB-IR sensor data.</p>	<p>At least one of the three --out0 <format> or --out1 <format> or -out2 <format> must be present.</p>
--out0size <wxh>	Specifies the downscale output0 size in the format of <wxh>.	If argument is not given, no downscaling occurs.
--out1size <wxh>	Specifies the downscale output1 size in the format of <wxh>.	If argument is not given, no downscaling occurs.
--out2size <wxh>	Specifies the downscale output2 size in the format of <wxh>.	If argument is not given, no downscaling occurs.
--out0crop <x+y+wxh>	Specifies the cropping of output0 size after downscaling.	If argument is not given, no cropping occurs.
--out1crop <x+y+wxh>	Specifies the cropping of output1 size after downscaling.	If argument is not given, no cropping occurs.
--out2crop <x+y+wxh>	Specifies the cropping of output2 size after downscaling.	If argument is not given, no cropping occurs.
--save-metadata	Saves metadata to file for each pipeline.	-
--header-skip <s1,s2>	Skips <s1> bytes in the file header and <s2> bytes in each frame header.	If option not specified, <s1> and <s2> are both 0.
--plug-in <type>	<p>Plugin used for control algorithm components. Accepts:</p> <ul style="list-style-type: none"> NV: NVIDIA AE/AWB plug-in SAMPLE: Sample plugin CUSTOM: Custom plugin 	NV
--nito <file>	Specifies NVIDIA ISP tuning object <file> to use.	Required argument.

6.7.5.2.4.3 Saving Metadata to a File

To save metadata to a file, add the --save-metadata argument when running nvsipl_reprocess. The metadata file gets saved in the format <prefix>_<isp>_cam_<id>_meta.txt.

Example: Saving Metadata from a V623S2-195V1-SVS Module in Two-lane CPHY Mode

```
$ ./nvsipl_reprocess -c "V623S2-195V1-SVS_CPHY_x2" -m "0000 0001 0000 0000" -i
input_file.raw --out0 yuv420_8b --save-metadata --nito V623S2-195V1-SVS.nito -f
"output"
```

Sample metadata output:

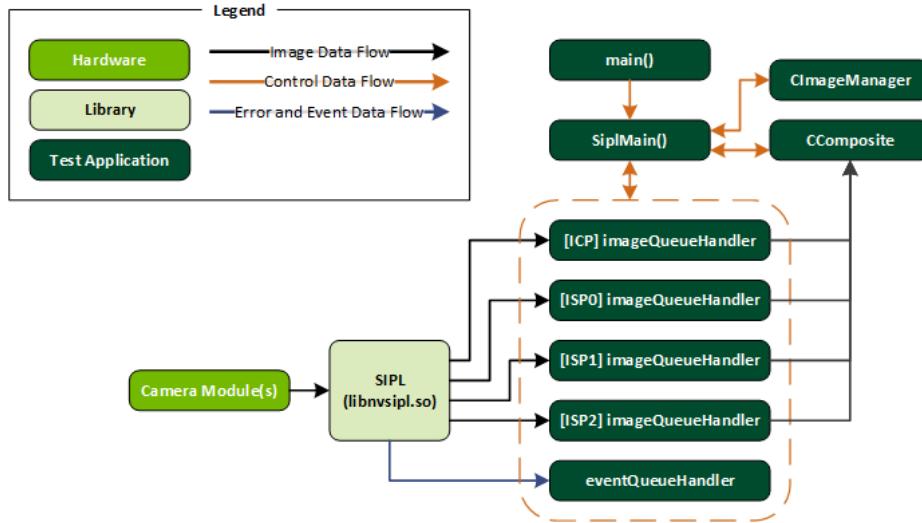
```
Camera ID: 12
Number of exposures: 3
Sensor exposure block valid: TRUE
Exposure Time T1: 0.017088
Exposure Time T2: 0.001034
Exposure Time T3: 0.000067
Sensor gain block valid: TRUE
Sensor Gain T1: 2.900000
Sensor Gain T2: 2.900000
Sensor Gain T3: 2.900000
White balance block valid: TRUE
White Balance Gain T1 for:
R component: 1.500000
Gr component: 1.000000
Gb component: 1.000000
B component: 1.500000
White Balance Gain T2 for:
R component: 1.500000
Gr component: 1.000000
Gb component: 1.000000
B component: 1.500000
White Balance Gain T3 for:
R component: 1.500000
Gr component: 1.000000
Gb component: 1.000000
B component: 1.500000
Sensor PWL block valid: FALSE
CRC block valid: FALSE
Frame Report block valid: FALSE
Temperature block valid: FALSE
```

6.7.5.2.5 SIPL Sample (nvsipl_sample)

The NvMedia nvsipl_sample sample application demonstrates how to use the NvMedia SIPL APIs to capture and process images from a camera module. The sample uses the INvSIPLCamera and INvSIPLClient interfaces to program the external image devices and the NVIDIA® Tegra® VI and ISP hardware engines. For more information about the NvSIPL APIs, see the API documentation for [NvSIPL](#).

6.7.5.2.5.1 Architecture

The following diagram shows the architecture of the `nvsipl_sample` application.



The test application executes the `SiplMain()` function in which it uses the `INvSIPLCamera` interface for communicating with the NvS IPL library and uses the `CIImageManager` class for buffer allocation and deallocation.

The platform configuration used by the application is fixed; all output types are enabled for one image processing pipeline. This pipeline captures and processes images from a single camera module.

The calibration data for the image processing pipeline is provided in the form of an NVIDIA Image Tuning Object (NITO) file in the target file system. NITO files are binary files that contain ISP settings, tuning settings, and characterization parameters for a specific camera module.

The main thread, executing the `SiplMain()` function, initializes the image processing pipeline using the NvS IPL APIs and then spawns a series of different threads. One thread is created for receiving the errors and events from the NvS IPL library and printing them to the command line. Other individual threads are created for receiving the output buffers for each of the output types of the pipeline. All these threads receive their associated items (either events or image buffers) by retrieving these items from queues provided by the NvS IPL library.

The received buffers contain an image and its associated metadata. Each image output thread reads the frame sequence number from the image metadata structure and prints it to the command line, then releases the buffer for subsequent capture and processing operations.

6.7.5.2.5.2 Running the Application

Before running `nvsipl_sample` it is necessary to follow the procedures in [Building and Running the NvMedia Samples](#).

To run the sample on the target:

1. Change the current directory to the folder that contains the nvsipl_sample binary:

```
/samples/nvmedia/nvsipl/test/sample/
```

2. Enter this command to launch the application:

```
$ ./nvsipl_sample -p LI-OV2311-VCSEL-GMSL2-60H-DPHY_x4
```

6.7.5.2.5.3 Command Line Switches

Option	Description
-p <platformCfgName>	<p>Specifies a platform configuration.</p> <p>Example:</p> <ul style="list-style-type: none"> > LI-OV2311-VCSEL-GMSL2-60H-DPHY_x4 > Refer to Camera Commands for the full list of supported configurations.
-v <level>	<p>Set verbosity</p> <p>Supported values (0,4)</p> <p>None - 0 Errors - 1 Warnings and above - 2 Info and above - 3 Debug and above - 4</p>
-d	Enable Display (QNX only)

6.7.5.2.5.4 Interactive Menu Options

Option	Description
q	Quits the application

6.7.5.2.5.5 Camera Commands and Platform Configs

This section describes supported camera commands for different platforms.

List of commands

Command	Description	Camera Group	No. of Cams	NVIDIA DRIVE Orin™ Developer Kit (P3710) Supported?	NVIDIA DRIVE Orin™ (P3663) Supported?
<code>./nvsipl_samp -p LI-OV2311-VCSEL-GMSL2-6DPHY_x4</code>	Leopard OV2311 (Camera Module LI-OV2311-VCSEL-GMSL2-60H) in Four Lane DPHY Mode	AB	1	Y	Y
<code>./nvsipl_samp -p V1SIM728S1RU3</code>	Valeo IMX728 B2 (Camera Module V1SIM728S1RU3) in Four Lane CPHY Mode	AB	1	Y	Y
<code>./nvsipl_samp -p V1SIM728S2RU3</code>	Valeo IMX728 B3 (Camera Module V1SIM728S2RU3) in Four Lane CPHY Mode	AB	1	Y	Y
<code>./nvsipl_samp -p V1SIM728S2RU4</code>	Valeo IMX728 B2.2 (Camera Module V1SIM728S2RU4) in Four Lane CPHY Mode	AB	1	Y	Y
<code>./nvsipl_samp -p V1SIM623S3RU3</code>	Valeo IMX728 B2 (Camera Module V1SIM623S3RU3) in Four Lane CPHY Mode	AB	1	Y	Y
<code>./nvsipl_samp -p V1SIM623S3RU3</code>	Valeo IMX623 B2 (Camera Module V1SIM623S3RU3) in Four Lane CPHY Mode	CD	1	Y	Y
<code>./nvsipl_samp -p V1STM623S4RL</code>	Valeo IMX623 B3.2 (Camera Module V1STM623S4RL) in Four Lane CPHY Mode	CD	1	Y	Y
NVIDIA DRIVE OS Linux SDK Developer Guide V1SIM623S4RU3				PR-10720-6.0_v6.0.8.1 636	

6.7.5.2.5.6 Camera Input Module (CIM) SKU Identification

To obtain the SKU information of a CIM ROM, execute the following command in the Aurix shell:

```
NvShell>cimrom get sku
```

For example, for a board with CIM SKU of 0002, the output from running the command is:

```
699-63553-0002-100
```

The last three digits may vary.

6.7.5.3 NvMedia IDE - Decode Processing (nvm_ide_sci)

NvMedia IDE - Decode Processing (nvm_ide_sci)

This topic describes how to use the NvMedia IDE - Decode Processing sample application, `nvm_ide_sci`, for stream decoding. The application supports these stream types:

- > MPEG
- > MPEG-4
- > VC1
- > H.264
- > H.265
- > VP8
- > VP9
- > AV1 (From T234 NVIDIA Orin™ and higher chips only)

Profiles and Supported Levels

Codec	Profile and level	Max video format, max bitrate	Comments
H.264	Baseline, Main, High Level 5.1 up to 8 bpp Level 5.1 DRM Support	4K@60, 120 Mbps @ 540 MHz	Exceptions as listed in IAS Max resolution: ➢ H265/VP9/AV1: 8192x8192 pixels ➢ H264: 4096x4096 ➢ MPEG2: 4080x4080 ➢ VP8: 4096x4096 ➢ VC1: 1920x1088 ➢ MPEG4: 1920x1088 Min resolution: ➢ H264: 48x48 ➢ H265: 144x144 ➢ AV1/VP9: 128x128 ➢ VP8/MPEG2/MPEG4/VC-1: Width >= 128 Perf mentioned is for 4:2:0, 8/10bpp ➢ Perf HALVED for 4:4:4 ➢ 8K = 7680*4320
HEVC	Main, Main10, Main12, Main444 12, MV Main Level 6.0 DRM Support	4K@120, 8K@30, 240 Mbps @800 MHz	New design, based on AV1 IP
VP8	All Profiles	4K@60, 120 Mbps @ 540 MHz	
VP9	Profile 0, Profile 2 Level 5.1 DRM Support	4K@120, 8K@30, 160 Mbps @800 MHz	New design, based on AV1 IP

Codec	Profile and level	Max video format, max bitrate	Comments
MPEG-2	SP, MP	4K@60, 120 Mbps	
	DRM Support	@ 540 MHz	
MPEG-4	SP, ASP	1080p@240, 120 Mbps @ 540 MHz	
VC1	SP, MP, AP	4K@220, 60 Mbps	
	Level 4	@ 540 MHz	
	DRM Support		
AV1	Main Profile	4K@120, 8K@30, 120 Mbps	
	Level 6.0		
	Bpp: 8/10	@800 MHz	
	Color: 4:2:0, 4:0:0 only	Max res-8kx8k	
	DRM Support		
	No support for High/Professional profiles or YUV4:2:2/YUV4:4:4		

6.7.5.3.1 Command Line Switches

The application's command syntax is:

```
./nvm_ide_sci [switches]
```

Required Commands

Required Switches

The following table describes the required command line switches:

Switch	Description
-c <codec_type>	<p>Specifies the codec type. You can specify the type with an integer or a string:</p> <ul style="list-style-type: none"> > 1 or mpeg: MPEG > 2 or mpeg4: MPEG-4 > 3 or vc1: VC1 > 4 or h264: H.264 > 5 or vp8: VP8 > 6 or h265: H265 > 7 or vp9: VP9 > 8 or av1 : AV1 (Orin T234 and higher chips only)
-f <input_file>	Specifies the input file to decode.

Other Switches

The following table describes the optional command line switches:

Switch	Description	Default
-t	Displays decode timing information. Default: Not displayed.	Information is not displayed.
-n <frames>	Specifies the minimum number of frames to decode. In some cases, however, specifying this option might show a false CRC mismatch issue toward the end of the frame number specified using the -n option.	Decodes the whole stream.
-s <output_file>	Specifies the output YUV file name.	No file is written.
-operating_point	AV1 Decode specific option for SVC streams only , for choosing operating point	0
-output_all_layers	AV1 Decode specific options for SVC streams, to Display all Decoded frames or only Highest Layer Enhancement layer.	1

Switch	Description	Default
-id <instance_id>	Specifies the nvdec instance ID to use for decoding. The value can be 0 or 1.	0
-annexBStream	AV1 codec-specific option to indicate that the input Bitstream is in AnnexB format.	0
-a <aspect_ratio>	Specifies the video stream's aspect ratio.	N/A
-v <level>	Specifies the logging level. The value can be: ➢ 0: Errors ➢ 1: Warnings ➢ 2: Info ➢ 3: Debug	0
-h	Prints help text for the application.	
-crc <gen/chk>	Enable CRC checks (chk) or generate CRC file (gen)	False
-crcpath <path>	Path for crc of the picture	Null
-cropcrc	CRC will be calculated on actual resolution	False

6.7.5.3.2 Examples

The nvm_ide_sci application can be used for stream decoding. Decoded frames can be saved to an output YUV file. This section provides several usage scenarios.

To perform decoding without saving decoded output

Enter this command:

```
./nvm_ide_sci -c 4 -f disney.264
```

The command starts decoding the disney.264 H.264 stream and does not save the output YUV to a file.

To perform decoding and save decoded output

Enter this command:

```
./nvm_ide_sci -c 4 -f disney.264 -s disney_out.yuv
```

The command starts decoding the disney.264 H.264 stream and saves the decoded frames to the `disney_out.yuv` file.

To specify the number of frames to decode

To specify the number of frames to decode, enter this command:

```
./nvm_ide_sci -c 4 -f disney.264 -n 100
```

This example decodes the first 100 frames from the `disney.264` file without saving the output to a file.

To specify the number of times to decode the frames, enter this command:

```
./nvm_ide_sci -c 4 -f disney.264 -l 10
```

This example decodes `disney.264` ten times.

To generate CRC for decoded YUV

Enter this command:

```
./nvm_ide_sci -c 4 -f disney.264 -crc gen -crcpath disney_PitchLinear_Ref_CRC.txt
```

This example decodes the stream `disney.264` and generates CRC of the decoded YUV, and then saves it in the file `disney_PitchLinear_Ref_CRC.txt`.

To check CRC for decoded YUV against a reference CRC

Enter this command:

```
./nvm_ide_sci -c 4 -f disney.264 -crc chk -crcpath disney_PitchLinear_Ref_CRC.txt
```

This example decodes the stream `disney.264`, computes CRC of the decoded YUV, and compares it with the reference CRC `disney_PitchLinear_Ref_CRC.txt`. It prints if the CRC matches the reference. If there is a mismatch, then the decoding is incorrect.

To get the decode timing information

Enter this command:

```
./nvm_ide_sci -c 4 -f disney.264 -prof
```

This example decodes the stream `disney.264` and prints information about the average time taken per frame and the total time taken for the sequence.

6.7.5.4 NvMedia IJPD - JPEG Decode (nvm_ijpd_sci)

The NvMedia IJPD application, `nvm_ijpd_sci`, demonstrates how to decode a set of JPEG bitstreams into uncompressed image surfaces using NvMedia IJPD APIs. It parses the command line with required parameter sets for the JPEG decoder, including the following:

- > Input bitstream file name

- > Output file name
- > Output resolution
- > Output format, etc.

It then reads the input bitstream file into the local buffer frame by frame and sends the information to the NvMedia IJPD. It saves the resulting image surfaces into the output YUV file.

The application also supports CRC generation and CRC checking modes. CRC generation is primarily for developers to provide a golden/reference CRC that can be used during verification. The CRC values can vary for different Tegra chip versions. CRC checking mode is for quality assurance to verify the application for new releases.

The following diagram describes the basic function flow.



6.7.5.4.1 Example Commands

To decode 100 JPEG files into 720 x 480 images in RGB format

Enter the command:

```
./nvm_ijpd_sci -f input%d.jpg -of outputfile -fn 100
```

To decode 100 JPEG files with CRC generation

Enter the command:

```
./nvm_ijpd_sci -f input%d.jpg -of outputfile -fn 100 -crcgen crc.txt
```

To decode 100 JPEG files with CRC checking

Enter the command:

```
./nvm_ijpd_sci -f input%d.jpg -of outputfile -fn 100 -crcchk crc.txt
```

To decode a monochrome (YUV400) file

Enter the command:

```
./nvm_ijpd_sci -f input.jpg -of outputfile -format 4
```

The default format value is 0 (yuv420). Format conversion from YUV400 to YUV420 is not supported. You must set the -format switch to 4 for monochrome decode.

Configuration Format

The -format parameter configures the color format of the decoded YUV. Supported formats are:

- > 0: yuv420 (the default)
- > 1: rgba
- > 2: yuv422
- > 3: yuv444
- > 4: yuv400

The output format that can be configured for a JPEG input depends on the format of the JPEG:

JPEG Format	Supported Output Formats
YUV444 JPEG	All the output formats are supported.
YUV422 JPEG	The following output formats are supported: <ul style="list-style-type: none"> > 0: yuv420 > 1: rgba > 2: yuv422
YUV420 JPEG	The following output formats are supported: <ul style="list-style-type: none"> > 0: yuv420 > 1: rgba
Unknown JPEG format	The application sets the output format to the default value of 0 (yuv420).
YUV400 JPEG (Monochrome)	The following output formats are supported: <ul style="list-style-type: none"> > 4: yuv400

Output YUV Resolution

The resolution of generated YUV is always aligned to the next multiple of 16. For example, if the input JPEG resolution is 800x600, the resolution of output YUV file is 800x608. Note that the image is not scaled from 800x600 to 800x608; the last eight lines must be ignored.

6.7.5.4.2 Command Line Switches

To run the sample application, enter this command:

```
./nvm_ijpd_sci [switches]
```

The following table describes the command line switches for the application:

Switch	Description	Default Settings
-h	Prints the help menu.	N/A
-f <file>	Specifies the input bitstream file. If the filename is <code>inputd.jpg</code> , the application tries to decode a set of input JPG files named <code>input1.jpg</code> , <code>input2.jpg</code> , etc. Otherwise, the single specified file's bitstream is decoded.	N/A
-of <file>	Specifies the YUV file.	N/A
-fr <w>x<h>	Specifies the output file resolution as width and height separated by an 'x', e.g. 640x480 or 1080x960.	N/A
-fn <value>	Specifies the decode frame number.	1
-format <value>	Specifies the output format. Values are: <ul style="list-style-type: none"> > 0: yuv420 > 1: rgba > 2: yuv422 > 3: yuv444 > 4: yuv400 	0
-ds <value>	Specifies the decode output downscale factor. The value can be [0,3].	0 (no downscale)
-crcgen <file>	Generates a CRC and saves it to the specified file.	N/A
-crcchk <file>	Generates a CRC and checks it against the specified CRC file.	N/A
-v <level>	Logging level. The value can be: <ul style="list-style-type: none"> > 0 (Errors) > 1 (Warnings) > 2 (Information) > 3 (Debug) 	If the switch is omitted, 0. If the switch is specified but <level> is omitted, 0.

6.7.5.5 Image JPEG Encode (nvm_ijpe_sci)

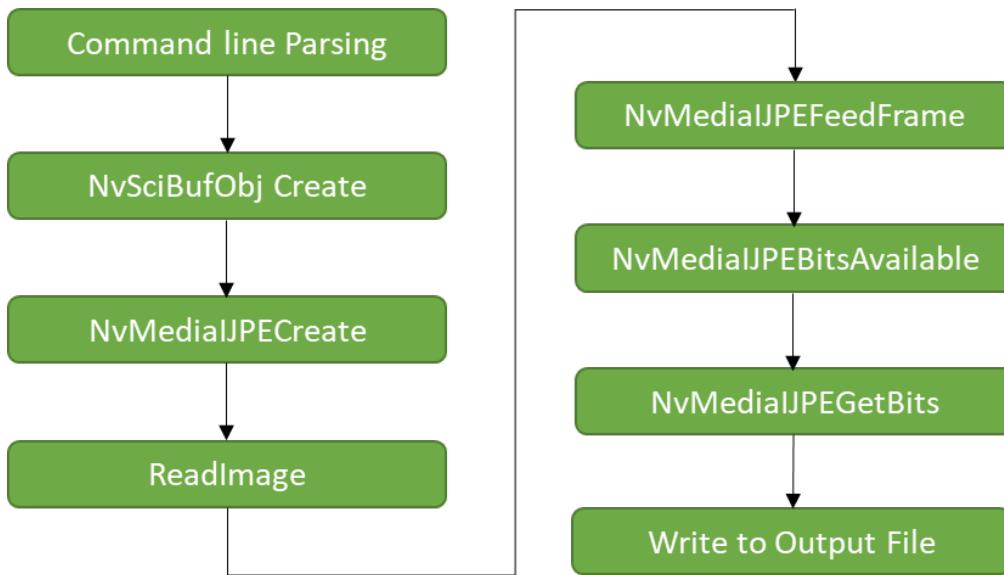
The NvMedia Image JPEG Encode application, `nvm_ijpe_sci`, demonstrates how to encode image surfaces into JPEG bitstreams using NvMedia JPEG encode APIs. It parses the command line with required parameter sets for JPEG encoder including:

- > Input file name
- > Resolution
- > Output bitstream file name
- > Quality, etc.

It then reads the input file into the image surface and sends the information to the NvMedia JPEG encoder. It saves the resulting bitstreams into a set of output files.

The application also supports CRC generation and CRC checking modes. CRC generation is mainly for developers to provide an original CRC. It can be a different CRC for different Tegra chip versions. CRC checking mode is for QA to verify the application for new releases.

The following diagram describes the basic function flow:



6.7.5.5.1 Example Commands

To encode a set of 720 x 480 JPEG files with quality 75

- > Enter this command:

```
./nvm_ijpe_sci -f input.yuv -fr 720x480 -of output%d.jpg -q 75
```

To encode with CRC generation

- > Enter this command:

```
./nvm_ijpe_sci -f input.yuv -fr 720x480 -of output%d.jpg -q 75 -crcgen crc.txt
```

To encode with CRC checking

- Enter this command:

```
./nvm_ijpe_sci -f input.yuv -fr 720x480 -of output%d.jpg -q 75 -crcchk crc.txt
```

6.7.5.5.2 Command Line Switches

To run the Image Encode sample application, enter this command:

```
./nvm_ijpe_sci [switches]
```

This table describes the command line switches:

e	Description	Default
-h	Prints the help menu.	N/A
-f <file>	Specifies the pathname of an input YUV file. The input file should be in YUV420 with UV order.	N/A
-fr <W>x<H>	Specifies the input file resolution in width × height format, for example: 640x480.	N/A
-of <file>	Specifies the pathname of an output file. If outputd.jpg is specified, the application generates a set of JPG files as output1.jpg, output2.jpg, etc. Otherwise, the last JPEG bitstream is saved to the specified file.	N/A
-q <value>	Specifies the encode quality. The value must be in the range [1,100].	50
-crcgen <file>	Generates a CRC and saves it to the specified file.	N/A
-crcchk <file>	Generates a CRC and checks it against the specified CRC file.	N/A
-v <level>	Logging level. Value may be: <ul style="list-style-type: none"> ➢ 0 (Errors) ➢ 1 (Warnings) ➢ 2 (Information) ➢ 3 (Debug) 	If the switch is omitted, 0. If the switch is specified but <level> is omitted, 3.

e	Description	Default
-HuffTable <file>	Specifies the pathname of a configuration file containing a Huffman table.	N/A
-QuantTable	Specifies the pathname of a configuration file containing a quant table.	N/A
-hwid	Specifies the hardware instance ID to be used.	0

6.7.5.6 NvMedia IEP – Encode Processing (nvm_iep_sci)

The NvMedia IEP sample application nvm_iep_sci demonstrates how to use the NvMedia IEP API to encode H.264/H.265/AV1 bitstreams with the NVENC hardware engine.

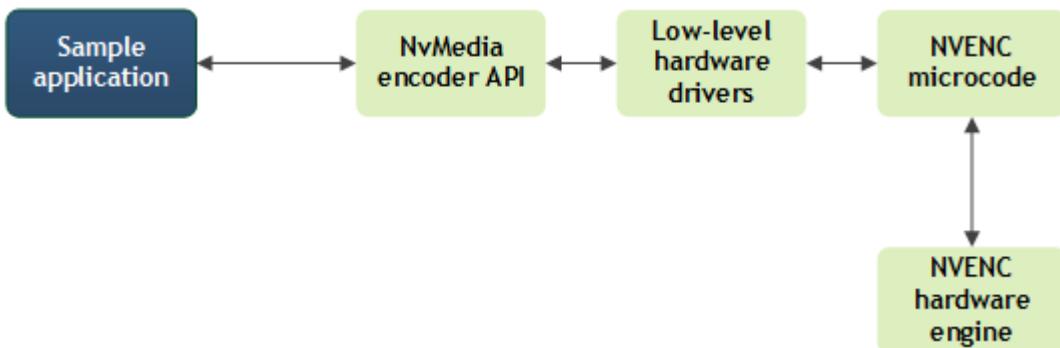
The application consumes raw images in a single YUV file in the YUV color space. It produces compressed elementary streams in one of the following formats:

- > H.264 with various configuration parameters
 - Baseline up to Level 5.2
 - Main profiles up to Level 5.2
 - High profiles up to Level 5.2
- > H.265 main profile up to level 6.0 with various configuration parameters
- > AV1 main profile up to level 6.3

nvm_iep_sci can run in event data recorder mode for a given number of seconds, as specified on the command line. The specified number of seconds of bitstream is saved to the output bitstream file.

6.7.5.6.1 Architecture and Data Flow

nvm_iep Stack Diagram



Component Descriptions

Name	Description
nvm_iep_sci	The sample application which calls the NvMedia IEP APIs. On receipt of the headers and bitstream, packetizes the encoded stream and saves the result to an output file.
NvMedia IEP API	Calls the low-level hardware driver interface to issue encoding commands to NVENC microcode.
Low-level hardware driver	<p>Writes out high level headers, including SPS, PPS, VUI and SEI.</p> <p>The driver creates and maintains reference picture surfaces, the output bitstream buffers, and the internal status buffers.</p> <p>For each input picture, the driver provides a configuration data structure (a picture setup) that contains encoder parameters along with the encode commands to the NVENC microcode. This structure contains:</p> <ul style="list-style-type: none"> > Buffer information (current picture, reference pictures) > Sequence and picture parameter set information and rate control configuration > Other module configurations as needed
NVENC microcode	Accesses the hardware engine to schedule the encoding commands, and return the encode status including bitstreams, statistics, etc.
NVENC hardware engine	Handles slice header and data.

NvMedia Low-Level Encode Driver Inputs and Outputs



Picture buffers must be in block linear format. Pitch linear input data is not supported due to inefficiency in the memory accesses.

The picture dimensions of the encoded streams are always in multiples of 4 pixels.

6.7.5.6.2 Running the Sample Application

The nvm_iep_sci application supports an input file of type YUV.

To run the application

1. Prepare the input file.
 - > The input is a single YUV file.
 - > To encode YUV files, the InputFile parameter must specify the pathname on the command line or in a configuration file.
2. Create a configuration for the encoding process.
3. Follow the steps in [Building and Running the NvMedia Samples](#)
4. Launch the application.

6.7.5.6.3 Command Line Switches

The format of the nvm_iep_sci application command line is:

```
nvm_iep_sci [-h] [-v] [-cf config.cfg] {[-sf config1.cfg]...[-sf configN.cfg]} {[ -p
EncParam1=EncValue1]...[ EncParamM=EncValueM]} -eventDataRecorder [num_second]
```

The following table describes the command line switches:

Switch	Description	Default
-h	Displays help text on for this application.	N/A
-v <level>	Logging level. Value may be: <ul style="list-style-type: none"> > 0: Errors > 1: Warnings > 2: Info > 3: Debug 	0
-cf <config_file>	Specifies the base configuration file.	N/A
-sf <config_file>	Specifies a specific configuration file. Any parameters provided in this file overwrite parameters specified in the base configuration file.	N/A

Switch	Description	Default
<code>-p <p1>=<v1>... <pn>=<vn></code>	<p>Sets parameter <code><p1></code> to the value <code><v1></code> ... <code><pn></code> to value <code><vn></code>.</p> <p>Overrides parameters set through both configuration files.</p> <p>Note: Invalid parameters keys are ignored.</p>	N/A
<code>-eventDataRecorder <sec></code>	Enables event data recorder mode with a recording time of <code><sec></code> seconds.	
<code>-crc gen <filename></code>	Generates CRC values for each frame in the encoded bitstream and stores them in the specified file.	N/A
<code>-crc chk <filename></code>	Generates CRC values at runtime for each frame in the encoded bitstream and checks them against the CRC values specified in the file.	N/A
<code>-id <id></code>	Specifies encoder instance ID.	0
<code>-profile <FPS></code>	Enables profiling measurement and reports Initialization, Submission, and Execution latency. <code><FPS></code> indicates the FPS at which Frames should be rate limited. 0 indicates rate limiting is disabled.	N/A
<code>-profileTest <InitLat> <SubLat> <ExecLat></code>	Verifies whether Initialization, Submit, and Execution latencies are within the prescribed limits when profiling is enabled. Where <code><InitLat></code> , <code><SubLat></code> and <code><ExecLat></code> indicates the init, submit and Execution latency reference values in us.	N/A

6.7.5.6.4 Examples for Performing Common Tasks

To encode to H.264/H265/VP9/AV1

Enter the command:

```
./nvm_iep_sci -cf enc_h264_sample.cfg
./nvm_iep_sci -cf simple_h265_cif.cfg
```

```
./nvm_iep_sci -cf enc_av1_sample.cfg
```

The encoding is performed using parameters supplied in the configuration file.

The base configuration file specifies the input and output file information, but it is more convenient to use a specific configuration file (with the -sf option) for these parameters or pass them through the command line using -p option as follows.

```
./nvm_iep_sci -cf enc_h264_sample.cfg -p FramesToBeEncoded=10
```

To run in event data recorder mode

Enter the command:

```
./nvm_iep_sci -cf enc_h264_sample.cfg -eventDataRecorder <sec>
```

In event data recorder mode, the application saves a specified number of seconds of bitstream in the output bitstream file. The number of seconds is specified by the eventDataRecorder option.

In event data recorder mode, the application forces the following encode parameters:

- > GOP size = frame rate, which equals to 1 second encode time
- > IDR period = GOP size
- > H264RepeatSPSPPSMode = 2, SPS/PPS is repeated for every IDR frame[JS1]

The settings guarantee that each GOP bitstream is decoded independently. The GOP size and IDR period can be set to half second encode time, or other values. The user application can choose values which fits the use case as desired.

To use a specific configuration file

Enter the command:

```
./nvm_iep_sci -cf enc_h264_sample.cfg -sf enc_h26x_specific_config.cfg
```

The encoding is performed using parameters supplied in the enc_h264_sample.cfg and enc_h26x_specific_config.cfg files. If a parameter is present in both files, the value in the specific configuration file enc_h26x_specific_config.cfg is used.

To enable profiling

Enter the command:

```
./nvm_iep_sci -cf enc_h264_sample.cfg -sf enc_h26x_specific_config.cfg -profile <FPS>
```

The "-profile" option dumps the latency information. A non-0 value for the <FPS> parameter indicates rate limiting.

6.7.5.6.5 Configuration File Syntax

When you run nvm_iep_sci you must provide a configuration file. A default configuration is available at:

```
<top>/drive-linux/samples/nvmedia/img_enc/enc_h264_sample.cfg
```

The configuration file contains parameters with values, groups or sections of parameters, and optional comments.

Parameters

Each parameter must be on a separate line. A parameter line contains the following tokens:

- The parameter name
- An equal sign (“=”)
- A value
- An optional comment

Here is an example of a parameter with a comment:

```
InputFile = "configs/input.yuv" # File to encode
```

An unknown parameter in a configuration file does not cause a failure; the parameter is ignored. An ill-formed parameter line causes the application to fail.

Comments

A comment consists of a pound sign (a '#') and any following text to the end of the line. A comment may appear at the end of any parameter line, or on a line by itself.

Sections

A section is a named group of parameters, containing a specific set of parameters. For example, the group RC_Params contains rate control parameters.



Note:

Available section types include:

- EncodePic_Params: per-frame parameters
- EncodePicH264_Params: specific H.264 per-frame parameters
- Payload: different arrays of SEI payloads can be used for every picture in H.264 encoding
- EncodePicH265_Params: consists of specific H.265 per frame parameters
- EncodePicAV1_Params: specific AV1 per frame parameters
- PayloadH265: different arrays of SEI payloads can be used for every picture in H.265 encoding
- RC_Params: rate control parameters
- QP_Params: quantization parameters

Each section begins with a line in the form:

```
[<section name>]
```

The rest of the section consists of parameter values to be defined in the group. A specific set of parameters may be defined in each group, as shown in [Configuration File Parameters](#).

For example, these lines define a QP_Params group:

```
[QP_Params]
QPBSlice      = 30    # Quant. parameter for B slices (0-51) - qpInterB
QPISlice       = 28    # Quant. parameter for I Slices (0-51) - qpIntra
QPPSlice       = 28    # Quant. parameter for P Slices (0-51) - qpInterP
```

6.7.5.6.6 Configuration File Parameters

Parameters that specify the length of an array, such as `ExplicitFrameInveralPatternLength` or `ExplicitFrameInveralPattern`, do not need to be set or may be set to 0. In either case, the application uses the default array size of 1000.

General Configuration Parameters

The following table describes the general configuration parameters:

Parameter	Description
InputFile	Identifies the YUV file fed as input to the encoder.
InputFileFormat	<p>Input file format. The supported values are listed below. For Codec-specific details, refer to Input File Format.</p> <ul style="list-style-type: none"> > 0: IYUV > 1: YV12 > 3: IYUV444 > 4: IYUV420_10bit (LSB aligned) > 5: IYUV444_10bit (LSB aligned) > 6: IYUV420_10bit (MSB aligned) > 7: IYUV444_10bit (MSB aligned)
OutputFile	Specifies the output file where the encoded bitstream will be written by the application.
StartFrame	Specifies the start frame for encoding. Encoding for <code>StartFrame-1</code> is skipped. Default value is 0
FramesToBeEncoded	Number of frames to encode. Set to 0 for encoding all frames in input YUV

Parameter	Description
EPCodec	<p>Specifies the video codec type. The values can be:</p> <ul style="list-style-type: none"> > 0: H.264 codec (Default) > 1: H.265 codec > 2: Reserved > 3: AV1 codec
ExtradataFileName	<p>Specifies the path to the file in which the frame-wise extra data will get dumped, when -enableExtradata flag is enabled. The extradata includes information such as frame average QP, frame type, intra/inter MB or CTB counts, and so on.</p>
DynamicResolutionFileName	<p>Specifies the path to the configuration file for Dynamic Resolution Change (DRC) feature. Refer to the DRC section below for the expected structure of the DRC configuration file.</p>
DRCBufferRealloc	<p>Specifies whether DRC support should be enabled using the buffer reallocation approach. The values can be:</p> <ul style="list-style-type: none"> > 0: Buffer reallocation disabled (default) > 1: Buffer reallocation enabled <p>This parameter should only be used along with InputFileDRC parameter. Refer to the DRC Buffer Reallocation section below for more information.</p>
InputFileDRC	<p>Specifies the path to the input file after the DRC resolution change. This parameter should only be used when DRCBufferRealloc is set to 1. Refer to the DRC Buffer Reallocation section below for more information.</p>

Parameter	Description
QPDeltaMapFileName	Specifies the base name of the per-frame QP delta map files. Refer to the QP delta map section below for the expected structure of the QP delta map files.
BitrateFileName	Specifies the path to the configuration file for Dynamic Bitrate Change (DBC) feature. Refer to the DBC section below for the expected structure of the DBC configuration file.
ROIParamFileName	<p>Specifies the path to the configuration file for the ROI encode feature, which is only applicable for H.264 and H.265 codecs. Following is the expected structure of the ROI file. Each line of the file corresponds to ROI parameters for one frame and should be formatted using the following structure:</p> <ul style="list-style-type: none"> > NumROIRegions : Range is 0-8 > x0: Left X co-ordinate > y0: Top Y co-ordinate > x1: Right X co-ordinate > y1: Bottom Y co-ordinate > QPdelta: QP delta for Region <p>The previous five parameters need to be repeated for each region for the current frame.</p>

Encode Configuration Parameters

The following table describes the encode configuration parameters:

Parameter	Description
EPEncodeWidth	Specifies the width of the encode.
EPEncodeHeight	Specifies the height of the encode.
EPFrameRateNum	Specifies the numerator and denominator for the frame rate to use for encoding, in frames per second. The frame rate is:
EPFrameRateDen	$EPFrameRateNum / EPFrameRateDen$
EPGopLength	Specifies the number of pictures in a GOP. If 0, keyframes are not inserted automatically.

Parameter	Description
EPGopPattern	<p>Specifies the GOP pattern. If the GOP length is 0, the Frame Interval Pattern must be set to IPP. The values can be:</p> <ul style="list-style-type: none"> > 0: I > 1: IPP > 2: IBP > 3: IBBP
EPMaxNumRefFrames	<p>Specifies the maximum number of reference frames. The value can be:</p> <ul style="list-style-type: none"> > EPMaxNumRefFrames =0 for I Only mode > EPMaxNumRefFrames =1 for IP mode > EPMaxNumRefFrames =2 for IBP mode
EPEnableROISlice	<p>Enables ROI encode by taking input region based deltaQp to be provided through ROIParamFileName. This is only applicable for H.264 and H.265 codecs.</p> <ul style="list-style-type: none"> > EPEnableROISlice=0 disables ROI encode > EPEnableROISlice=1 enables ROI encode

Quantization Configuration Parameters

The following table describes the quantization configuration parameters:

Parameter	Description
QPBSlice	Specifies the Quantization parameter for B slices .
QPISlice	Specifies the Quantization parameter for I Slices.
QPPSlice	Specifies the Quantization parameter P Slices.

The range of QP values supported for each of the above parameter is as follows:

H264/HEVC -[0-51]

AV1 - [1-255]

Rate Control Configuration Parameters

The following table describes the rate control configuration parameters:

Parameter	Description
RCMode	<p>Specifies the rate control mode. The values can be:</p> <ul style="list-style-type: none"> > 0x0 = Constant bitrate mode > 0x1 = Constant QP mode > 0x2 = Variable bitrate mode > 0x3 = Variable bitrate mode with MinQP
RCAverageBitrate	Specifies the average bit rate in bits/second; used for encoding.
RCMaxBitrate	Specifies the maximum bit rate for the encoded output. This is used in Variable Bit Rate (VBR) mode and is ignored for Constant Bit Rate (CBR) mode.
RCVbvBufferSize	Specifies the VBV (HRD) buffer size in bits. Set to 0 to use the default VBV buffer size.
RCVbvInitialDelay	Specifies the VBV (HRD) initial delay in bits. Set to 0 to use the default VBV initial delay.
RCEnableMinQP	Set to 1 if minimum QP is used for the rate control.
RCEnableMaxQP	Set to 1 if maximum QP is used for the rate control.

Per-Frame Encode Configuration Parameters

The following table describes the per-frame encode configuration parameters:

Parameter	Description
EPEncodePicFlags	<p>Specifies the bitwise OR'ed encode picture flags.</p> <p>The flags are:</p> <ul style="list-style-type: none"> > Ox1: Encodes the current picture as an Intra picture. > Ox2: Encodes the current picture as an IDR picture. This flag is valid when the encoder makes the picture type decision. (Set EPPictureType to 0). > Ox4: Writes the sequence and picture header in the encoded bitstream of the current picture. > Ox8: Indicates the end of the input stream. > Ox10: Indicates a change in bit rate from the current picture onwards. > Ox20: Indicates that the user forced constant QP Rate control from the current picture onwards. > Ox40: Indicates a change in the rate control mode on the fly from current picture onwards.
EPPictureType	<p>Specifies the input picture type. The client must set this parameter explicitly if it has not set the Enable PTD (picture type decision) to 1.</p> <p>The picture types are:</p> <ul style="list-style-type: none"> > Ox00: Forward predicted picture > Ox01: Bidirectionally predicted picture > Ox02: Intra predicted picture > Ox03: IDR picture > Ox04: Bidirectionally predicted picture with only Intra MBs > Ox05: Picture is skipped > Ox06: First picture in intra refresh cycle > OxFF: Picture type unknown

H.264 Encode Configuration Parameters

The following table describes the H.264 encode configuration parameters:

Parameter	Description						
H264Profile	<p>Supported profiles are:</p> <ul style="list-style-type: none"> > 0: Automatic profile selection > 66: Baseline profile > 77: Main profile > 88: Extended profile > 100: High profile > 244: High444 profile 						
H264Level	<p>Specifies the encoding level. The recommendation is for the client to set the level to 0 to enable the NvMedia Encode interface to select the correct level.</p>						
H264EncPreset	<p>Specifies the encoder preset to use</p> <table> <tr> <td>0x0</td> <td>= NVMEDIA_ENCODE_QUALITY_HQ</td> </tr> <tr> <td>0x10</td> <td>= NVMEDIA_ENCODE_QUALITY_HP</td> </tr> <tr> <td>0x20</td> <td>= NVMEDIA_ENCODE_QUALITY_UHP</td> </tr> </table> <p>recommend quality setting is NVMEDIA_ENCODE_QUALITY_HP</p>	0x0	= NVMEDIA_ENCODE_QUALITY_HQ	0x10	= NVMEDIA_ENCODE_QUALITY_HP	0x20	= NVMEDIA_ENCODE_QUALITY_UHP
0x0	= NVMEDIA_ENCODE_QUALITY_HQ						
0x10	= NVMEDIA_ENCODE_QUALITY_HP						
0x20	= NVMEDIA_ENCODE_QUALITY_UHP						
H264Features	<p>Specifies the bitwise OR'ed configuration feature flags. The flags are:</p> <ul style="list-style-type: none"> > ENABLE_OUTPUT_AUD (1 << 0) > ENABLE_INTRA_REFRESH (1 << 1) > ENABLE_DYNAMIC_SLICE_MODE (1 << 2) > ENABLE_CONSTRANED_ENCODING (1 << 3) > ENABLE_LOSSLESS_COMPRESSION (1 << 4) 						
H264IdrPeriod	<p>Specifies the IDR interval. If not set, it defaults to the GOP length. A low latency application client must set the IDR interval to 0 so that IDR frames are not inserted automatically.</p>						
H264RepeatSPSPPSMode	<p>Specifies the frequency of writing Sequence and Picture parameters. The values can be:</p> <ul style="list-style-type: none"> > 0x0: Repeating SPS/PPS is disabled > 0x1: SPS/PPS is repeated for every intra frame > 0x2: SPS/PPS is repeated for every IDR frame 						
H264NumSliceCountMinus1	<p>One less than the number of slices desired per frame.</p>						
H264DisableDeblockingFilterIDC	<p>Deblocking filter mode. The value can be 0, 1, or 2.</p>						

Parameter	Description
H264IntraRefreshPeriod	<p>The interval between successive intra refreshes if intra refresh is enabled and one-time intra refresh configuration is desired.</p> <p>If H264IntraRefreshPeriod is specified, the first IDR is encoded, and no more key frames are encoded.</p> <p>The client must set EPPictureType to 6 for the first picture of every intra refresh period.</p>
H264IntraRefreshCnt	<p>The number of frames over which intra refresh occurs.</p>
H264MaxSliceSizeInBytes	<p>The maximum slice size, in bytes, for dynamic slice mode. The client must enable dynamic slice mode to use this parameter.</p>
H264AdaptiveTransformMode	<p>Specifies the Adaptive Transform Mode. Available modes are:</p> <ul style="list-style-type: none"> ➢ 0x0: The encoder driver automatically selects Adaptive Transform 8x8 mode. ➢ 0x1: Adaptive Transform 8x8 mode disabled. ➢ 0x2: Adaptive Transform 8x8 mode must be used.

Parameter	Description
H264BdirectMode	<p>Specifies the B Direct mode. Available modes are:</p> <ul style="list-style-type: none"> ➢ 0x0: Spatial B Direct mode ➢ 0x1: Disable B Direct mode ➢ 0x2: Temporal B Direct mode
H264EntropyCodingMode	<p>Specifies the entropy coding mode. Available modes are:</p> <ul style="list-style-type: none"> ➢ 0x0: The encoder driver automatically selects the entropy coding mode ➢ 0x1: Entropy coding mode is CABAC ➢ 0x2: Entropy coding mode is CAVLC
H264MotionPredictionExclusionFlags	<p>Specifies the bitwise OR'ed exclusion flags for motion prediction. Available flags are:</p> <ul style="list-style-type: none"> ➢ (1 << 0): Disable Intra 4x4 vertical prediction ➢ (1 << 1): Disable Intra 4x4 horizontal prediction ➢ (1 << 2): Disable Intra 4x4 DC prediction ➢ (1 << 3): Disable Intra 4x4 diagonal down left prediction ➢ (1 << 4): Disable Intra 4x4 diagonal down right prediction ➢ (1 << 5): Disable Intra 4x4 vertical right prediction ➢ (1 << 6): Disable Intra 4x4 horizontal down prediction ➢ (1 << 7): Disable Intra 4x4 vertical left prediction ➢ (1 << 8): Disable Intra 4x4 horizontal up prediction ➢ (1 << 9): Disable Intra 8x8 vertical prediction ➢ (1 << 10): Disable Intra 8x8 horizontal prediction ➢ (1 << 11): Disable Intra 8x8 DC prediction ➢ (1 << 12): Disable Intra 8x8 diagonal down left prediction ➢ (1 << 13): Disable Intra 8x8 diagonal down right prediction ➢ (1 << 14): Disable Intra 8x8 vertical right prediction ➢ (1 << 15): Disable Intra 8x8 horizontal down prediction ➢ (1 << 16): Disable Intra 8x8 vertical left prediction ➢ (1 << 17): Disable Intra 8x8 horizontal up prediction ➢ (1 << 18): Disable Intra 16x16 vertical prediction ➢ (1 << 19): Disable Intra 16x16 horizontal prediction

H.264 VUI Configuration Parameters

The following table describes the H.264 VUI configuration parameters:

Parameter	Description
VUIAspectRatioInfoPresentFlag	A value of 1 indicates that the aspect ratio information is present.
VUIAspectRatioIDC	Specifies the sample aspect ratio of the luma samples.
VUIAspectSARWidth	Specifies the horizontal size of the sample aspect ratio.
VUIAspectSARHeight	Specifies the vertical size of the sample aspect ratio.
VUIOverscanInfoPresentFlag	A value of 1 indicates that overscan info is present.
VUIOverscanInfo	Specifies overscan information, as defined in Annex E of the ITU-T Specification.
VUIVideoSignalTypePresentFlag	If set to 1, specifies that Video Format, Video Full Range Flag, and Color Description Present Flag are present.
VUIVideoFormat	Specifies the source video format as defined in Annex E of the ITU-T Specification.
VUIVideoFullRangeFlag	Specifies the output range of the luma and chroma samples, as defined in Annex E of the ITU-T Specification.
VUIColourDescriptionPresentFlag	A value of NVMEDIA_TRUE indicates that the color primaries, transfer characteristics, and color matrix are present.
VUIColourPrimaries	Specifies the color primaries for converting to RGB, as defined in Annex E of the ITU-T Specification.
VUITransferCharacteristics	Specifies the opto-electronic transfer characteristics to use, as defined in Annex E of the ITU-T Specification.
VUIMatrixCoefficients	Specifies the matrix coefficients used to derive the luma and chroma from the RGB primaries, as defined in Annex E of the ITU-T Specification.

H.264 Payload Configuration Parameters

The following describes the H.264 payload configuration parameters:

Parameter	Description
H264PayloadSize	SEI payload 1 size in bytes. SEI payload must be byte aligned, as described in Annex D of the H.264 Specification.
H264PayloadType	SEI payload 1 types and syntax is available in Annex D of the H.264 Specification.
H264Payload	Payload 1 data.

H.264 Per-Frame Encode Configuration Parameters

The following table describes the H.264 per-frame encode configuration parameters. These are in addition to the per-frame encode configuration parameters. These parameters must be sent on a per-frame basis.

Parameter	Description
H264PayloadArraySize	Size of Payload Array.
H264PayloadArrayIndexes	Array of the Payload section indexes to be used.

H.265 Configuration Parameters

The following table describes the H.265 configuration parameters:

Parameter	Description
H265Profile	Supported profiles are: 1: Main profile (supported)
H265Level	Specifies the encoding level. It is recommended that the client set the level to 0 to enable the NvMedia Encode interface to select the correct level.
H265EncPreset	Specifies the encoder preset to use 0x0 = NVMEDIA_ENCODE_QUALITY_HQ 0x10 = NVMEDIA_ENCODE_QUALITY_HP 0x20 = NVMEDIA_ENCODE_QUALITY_UHP recommend quality setting is NVMEDIA_ENCODE_QUALITY_HP

Parameter	Description
H265Features	<p>Specifies bitwise OR'ed configuration feature flags. The flags are:</p> <ul style="list-style-type: none"> ➢ <code>ENABLE_OUTPUT_AUD</code> (<code>1 << 0</code>) ➢ <code>ENABLE_INTRA_REFRESH</code> (<code>1 << 1</code>) ➢ <code>ENABLE_DYNAMIC_SLICE_MODE</code> (<code>1 << 2</code>) ➢ <code>ENABLE_CONSTRANED_ENCODING</code> (<code>1 << 3</code>) ➢ <code>ENABLE_LOSSLESS_COMPRESSION</code> (<code>1 << 4</code>) ➢ <code>ENABLE_SLICE_LEVEL_OUTPUT</code> (<code>1 << 5</code>)
H265IdrPeriod	<p>Specifies the IDR interval. If not set, defaults to the GOP Length. A low latency application client can set the IDR interval to 0 so that IDR frames are not inserted automatically.</p>
H265RepeatSPSPPSMode	<p>Specifies the frequency of writing Sequence and Picture parameters.</p> <p>Available values are:</p> <ul style="list-style-type: none"> ➢ 0x0: Repeating SPS/PPS is disabled ➢ 0x1: SPS/PPS is repeated for every intra frame ➢ 0x2: SPS/PPS is repeated for every IDR frame
H265NumSliceCountMinus1	<p>One less than the number of slices desired per frame.</p>
H265IntraRefreshPeriod	<p>The interval between successive intra refreshes, assuming:</p> <ul style="list-style-type: none"> ➢ Intra refresh is enabled. ➢ one time intra refresh configuration is desired. <p>If <code>H265IntraRefreshPeriod</code> is specified, the first IDR is encoded and no more key frames are encoded.</p> <p>The client must set <code>EPPictureType</code> to 6 for the first picture of every intra refresh period.</p>
H265IntraRefreshCnt	<p>Specifies the number of frames over which intra refresh occurs.</p>
H265MaxSliceSizeInBytes	<p>Specifies the maximum slice size, in bytes, for dynamic slice mode. The client must enable dynamic slice mode to use this parameter.</p>

H.265 VUI Configuration Parameters

The following table describes the H.265 VUI configuration parameters:

Parameter	Description
H265VUIAspectRatioInfoPresentFlag	A value of 1 indicates that aspect ratio information is present.
H265VUIAspectRatioIDC	Specifies the value of the sample aspect ratio of the luma samples.
H265VUIAspectRatioSARWidth	Horizontal size of the sample aspect ratio.
H265VUIOverscanInfoPresentFlag	A value of 1 indicates that overscan information is present.
H265VUIOverscanInfo	Specifies overscan information, as defined in Annex E of the ITU-T Specification.
H265VUIVideoSignalTypePresentFlag	A value of 1 indicates that Video Format, Video Full Range Flag, and Color Description Present Flag are present.
H265VUIVideoFormat	Specifies the source video format, as defined in Annex E of the ITU-T Specification.
H265VUIVideoFullRangeFlag	Specifies the output range of the luma and chroma samples, as defined in Annex E of the ITU-T Specification.
H265VUIColourDescriptionPresentFlag	A value of NVMEDIA_TRUE indicates that the color primaries, transfer characteristics, and color matrix are present.
H265VUIColourPrimaries	Specifies color primaries for converting to RGB, as defined in Annex E of the ITU-T Specification.
H265VUITransferCharacteristics	Specifies the opto-electronic transfer characteristics to use, as defined in Annex E of the ITU-T Specification.
H265VUIMatrixCoefficients	Specifies the matrix coefficients to use to derive the luma and chroma from the RGB primaries, as defined in Annex E of the ITU-T Specification.

H.265 Payload Configuration Parameters

The following table describes the H.265 payload configuration parameters:

Parameter	Description
H265PayloadSize	SEI payload 1 size in bytes. SEI payload must be byte aligned, as described in Annex D of the ITU-T Specification.
H265PayloadType	SEI payload 1 types and syntax are available in Annex D of the ITU-T Specification.
H265Payload	Payload 1 data.

H.265 Per-Frame Encode Configuration Parameters

The following table describes the H.265 per-frame encode configuration parameters. These parameters must be sent on a per frame basis.

Parameter	Description
H265PayloadArrayIndexes	Array of Payload section indexes to be used.
H265PayloadArraySize	Size of Payload Array.

AV1 Configuration Parameters

The following table describes the AV1 configuration parameters:

Parameter	Description
EPEnableTileEncode	Enable encoding with multiple tile configuration.
EPlog2NumTileInRow	Specifies the log of number of tiles in a row.
EPlog2NumTileInCol	Specifies the log of number of tiles in a column.
AV1EnableInternalHighBitDepth	Enables internal high bit depth. When enabled, input YUV is 8 bit, but the encoded bitstream is 10 bit.
AV1EnableSsimRdo	Enables SSIM-based rate distortion optimization
AV1FrameRestorationType	Specifies the frame restoration type.
AV1EnableUniCompound	Enables the uni-compound for P frames.

Parameter	Description
AV1IdrPeriod	<p>Specifies the IDR interval. If not set, the interval is equal to the GOP length. Low latency application client can set the IDR interval to 0 so that IDR frames are not inserted automatically.</p> <p>Default value = 0</p>
AV1EncPreset	<p>Specifies the encoder preset to use</p> <p>0x10 = NVMEDIA_ENCODE_QUALITY_HP 0x20 = NVMEDIA_ENCODE_QUALITY_UHP</p> <p>recommend quality setting is NVMEDIA_ENCODE_QUALITY_HP</p>
AV1Features	<p>Specifies bitwise OR'ed configuration feature flags. The flags are:</p> <ul style="list-style-type: none"> > ENABLE_QUANTIZATION_PARAMS (1 << 1), > DISABLE_CDF_UPDATE (1 << 2), > INIT_QP (1 << 6), > QP_MAX (1 << 7)

Supported Input File Format

H264 Enc supports the following File formats- IYUV, YV12, YUV444

H265 Enc supports the following File formats- IYUV, YV12, YUV444, IYUV420_10bit (LSB aligned), IYUV444_10bit (LSB aligned), IYUV420_10bit (MSB aligned), IYUV444_10bit (MSB aligned)

AV1 Enc supported the following File formats - IYUV, YV12, IYUV420_10bit (LSB aligned), IYUV420_10bit (MSB aligned)

Configuration Parameters for Dynamic Features

The following dynamic features can be configured using additional input files:

Dynamic Resolution Change (DRC)

This feature can be enabled by providing the DynamicResolutionFileName parameter, which should be set to the path of a file containing the resolution change information. Each line of the file should be formatted as per the following structure:

```
<frame_id> <width> <height>
```

For each line in the preceding configuration file, the specified resolution will take effect for all the frames starting from the specified frame_id until the frame_id specified on the next line (if any) or the end of the encode session, whichever occurs first.

The DRC feature can be applied in two different modes, as described below:

DRC without buffer reallocation

By default, the DRCBufferRealloc parameter is set to 0, to indicate that buffer reallocation is disabled.

- > The test app allocates all buffers at the maximum DRC resolution specified by EncodeWidth x EncodeHeight, and reads all the frames of the input YUV file at the same max resolution.
- > Within each input frame, the actual frame to be encoded should be positioned by aligning to the top-left corner, with the rest of the frame buffer being filled with black padding, to match the maximum DRC resolution.

DRC with buffer reallocation

If the DRCBufferRealloc parameter is set to 1, then the other parameters should be set as per the following requirements:

- > The DynamicResolutionFileName parameter should be configured as per the preceding description; and the specified file should contain only one line, because only one resolution change can be supported using the buffer reallocation approach.
- > The InputFile parameter should be used to identify the input YUV file for the initial resolution, whereas the InputFileDRC parameter should be used to identify the input YUV file after the resolution change.

If Buffer reallocation is configured as above, then the Dynamic Resolution Change feature is applied as per the following process:

- > The test app pre-allocates buffers for the initial resolution as well as the changed resolution after DRC.
- > The test app starts by reading frames from the InputFile for encoding at the initial resolution.
- > When the processing reaches the frame_id specified on the first (and only) line of DynamicResolutionFileName, the test app switches to using the buffers allocated for the changed resolution.
- > The test app now starts reading the subsequent frames from InputFileDRC file, until the end of the encode session.

Dynamic Bitrate Change

This feature can be enabled by providing the BitrateFileName parameter, which should be set to the path of a file containing the bitrate change information. Each line of the file should be formatted as per the following structure:

```
<frame_id> <bitrate_bps> <vbv_buffer_size_bits>
```

For each line in the above configuration file, the specified bitrate settings will take effect for all the frames starting from the specified frame_id until the frame_id specified on the next line (if any) or the end of the encode session, whichever occurs first.

QP delta map

This feature can be enabled by providing the QPDeltaMapFileName parameter, which should be set to the basename of a path containing input QP delta files for each frame. The filename of each frame's QP delta map should be as per the format `<QPDeltaMapFileName>_%05d.bin` where `%05d` will be replaced with the frame index starting from 1.

Usage guide for Dynamic Features

The following guidelines are provided as recommended best practices while using the above features; in particular to improve the rate control characteristics of the encoding. In this section, "dynamic parameter" refers to any parameter that can be modified using the preceding features; such as bitrate, resolution, and QP delta map.

1. After changing any dynamic parameter, there should be a gap of at least 3 frames before the same or different dynamic parameter is changed. This will allow the rate control algorithm to stabilize at the current operating point, and avoid bitrate overflows.
2. While applying DRC, the Encode Configuration Parameters `EPEncodeWidth`, `EPEncodeHeight` should be set to the maximum target resolution that would be applied during the encode session. In addition, the encoding should always start with this maximum target resolution; in other words, DRC should not be applied to the first frame of the session.
3. Although DRC supports changes from any initial resolution to any new target resolution, it is recommended to perform a gradual resolution change, by applying not more than 2x increase or decrease in the width and height of the resolution at a time. For example, to instead of changing the resolution from 540p(960x540) to 4k(3840x2160), a smoother rate control can be achieved by first switching from 540p to 1080p(1920x1080), and then after 3 frames to 4k.
4. The dimensions of QP delta map for each frame should be computed as follows:

```
#define ALIGN_256(x) (x << 8)
frameWidthInCTB = (EPEncodeWidth + CTB_SIZE - 1)/CTB_SIZE
frameHeightInCTB = (EPEncodeHeight + CTB_SIZE - 1)/CTB_SIZE
QPDeltaBufferSizeInBytes = ALIGN_256(frameWidthInCTB) * (frameHeightInCTB + 2)
```

where `CTB_SIZE=32`, and `EPEncodeWidth`, `EPEncodeHeight` are as per the Encode Configuration Parameters. Each entry in the QP delta map file should be a signed 8-bit integer, denoting the QP delta value for a CTB of 32x32 pixels. The frame should be scanned row-wise from the top-left to bottom-right.

5. In particular, the dimensions of the QP delta map buffer do not change even while applying Dynamic Resolution Change. In this case,, the QP delta map values should be arranged as if the actual frame to be encoded is positioned by aligning to the top-left corner, with the rest of the frame buffer being filled with black padding, to match the maximum DRC resolution.
6. QP delta map should not contain any values with a magnitude exceeding 5. In addition, the average value of QP delta map should be close to zero, for each row of CTBs within the frame. This allows a uniform allocation of the bits throughout the frame.
7. It is possible to update multiple dynamic parameters at the same frame; and it may also be necessary to take advantage of the same. For example, the target bitrate should be adjusted based on the current resolution, as well as the complexity of the encoding content. Increasing the resolution without increasing the target bitrate causes a reduction in video quality.

6.7.5.7 Image LDC (nvmimsg_ldc)

The NvMedia Image LDC sample application, `nvmimsg_ldc`, demonstrates how to use the NvMedia Lens Distortion Correction (LDC) APIs to perform Geometric Transformation (GEOTRANS) and Temporal Noise Reduction (TNR). The sample uses the NVIDIA® VIC hardware engine.

The application can read YUV files and perform GEOTRANS and TNR on input YUV files. It uses a configuration file that specifies various settings. It can also write the output surfaces as YUV files.

6.7.5.7.1 Running the Sample Application

This topic describes how to run the sample application.

To run the sample application

- Create a configuration file. You can use the `ldc_*.conf` files packaged in the platform as references.
- Follow the steps in [Building and Running the NvMedia Samples](#).
- Launch the application:

```
$ nvmimsg_ldc -cf /path/to/config/file.conf -v 3
```

6.7.5.7.2 Configuration File Parameters

Sample configuration files (.conf files) are available in platform install directory under:

```
samples/nvmedia/img_ldc/
```

A configuration file contains parameters with values, groups or sections of parameters, and comments.

6.7.5.7.2.1 General Configuration Parameters

This table describes the general configuration parameters:

Parameter	Description	Type and Range
<code>versionMajor</code>	Supported API version major number.	uchar: [3, 3]
<code>versionMinor</code>	Supported API version minor number.	uchar
<code>numFrames</code>	Number of frames to be processed.	uint: [1,f] where f is the number of frames in the input file]
<code>inputFile</code>	Name of input YUV file.	string: maximum 1024 characters

Parameter	Description	Type and Range
outputFile	Name of output YUV file.	string: maximum 1024 characters
checksumMode	Checksum calculation mode. Supported values are: 0: DISABLED 1: SRC_SURFACE	uint: [0, 1]
xSobelFile	Outputs an xSobel file.	string: maximum 1024 characters
xSobelDSFile	Outputs a 4×4 downsampled xSobel file.	string: maximum 1024 characters
srcWidth	Width of source surface.	ushort: [64, 16384]
srcHeight	Height of source surface.	ushort: [16, 16384]
srcRectx0	Left X coordinate of source rectangle.	ushort: [0, srcWidth]
srcRecty0	Top Y coordinate of source rectangle.	ushort: [0, srcHeight]
srcRectx1	Right X coordinate of source rectangle.	ushort: [0, srcWidth]
srcRecty1	Bottom Y coordinate of source rectangle.	ushort: [0, srcHeight]
dstWidth	Width of destination surface.	ushort: [64, 16384]
dstHeight	Height of destination surface.	ushort: [16, 16384]
dstRectx0	Left X coordinate of destination rectangle.	ushort: [0, dstWidth)
dstRecty0	Top Y coordinate of destination rectangle.	ushort: [0, dstHeight)
dstRectx1	Right X coordinate of destination rectangle.	ushort: [0, dstWidth)
dstRecty1	Bottom Y coordinate of destination rectangle.	ushort: [0, dstHeight)
enableGeotrans	Enables Geotrans processing	uint: [0, 1]

Parameter	Description	Type and Range
enableMaskMap	Enables Bitmask feature	uchar: [0, 1]
enableTnr	Enables TNR	uchar: [0, 1]
generateWarpMap	Enables warp map autogenerating based on the camera model.	uint: [0, 1]
applyWarpMap	Enables reading ready warp map from the file.	uint: [0, 1]
writeXSobel	Enables writing xSobel surface to the file	uint: [0, 1]
writeXSobelDS	Enables writing downsampled xSobel surface to the file	uint: [0, 1]

6.7.5.7.2.2 Geometric Transformation Parameters

This table describes the Geometric Transformation Parameters:

Parameter	Description	Range
filter	Filter quality. Supported values are: 0: LOW 1: MEDIUM 2: HIGH	uint: [0, 2]
model	Lens Model. Supported values are: 0: POLYNOMIAL 1: FISHEYE_EQIDISTANT 2: FISHEYE_EQUISOLID 3: FISHEYE_ORTHOGRAPHIC 4: FISHEYE_STEREOGRAPHIC	uint: [0, 4]

Parameter	Description	Range
k1 k2 k3 k4 k5 k6	Radial distortion coefficients.	float
p1 p2	Tangential distortion coefficients.	float
fx fy	Camera focal length in X and Y-axis, measured in pixel units.	float
cx cy	Camera optical center in X and Y-axis, measured in pixel units.	float
R00 R01 R02 R10 R11 R12 R20 R21 R22	Rotation matrix elements.	float
T0 T1 T2	Translation vector elements.	float

Parameter	Description	Range
targetKfx targetKfy	Target camera focal length in X and Y-axis, measured in pixel units.	float
targetKcx targetKcy	Target camera optical center in X and Y-axis, measured in pixel units.	float
ptMatrix00 ptMatrix01 ptMatrix02 ptMatrix10 ptMatrix11 ptMatrix12 ptMatrix20 ptMatrix21 ptMatrix22	Perspective matrix elements.	float
numHorRegion	Number of horizontal regions.	uint: [1, 4]
numVerRegion	Number of vertical regions.	uint: [1, 4]
horRegionWidth0 horRegionWidth1 horRegionWidth2 horRegionWidth3	Widths of regions.	uint
verRegionHeight0 verRegionHeight1 verRegionHeight2 verRegionHeight3	Heights of regions.	uint

Parameter	Description	Range
log2horSpace0 log2horSpace1 log2horSpace2 log2horSpace3	Horizontal interval between the control points in regions in log2 space.	uint
log2verSpace0 log2verSpace1 log2verSpace2 log2verSpace3	Vertical interval between the control points in regions in log2 space.	uint
bitMaskWidth	Width of the bitmask surface.	uint
bitMaskHeight	Height of the bitmask surface.	uint
bitMaskFile	Pathname of a file containing a bitmask map.	string: maximum 1,024 characters
maskedPixelFillColor	Indicates whether to fill the masked pixels with specified color.	uint: [0,1]
maskY	Y channel value of the default color.	float: [0.0,1.0]
maskU	U channel value of the default color.	float: [0.0,1.0]
maskV	V channel value of the default color.	float: [0.0,1.0]

6.7.5.7.2.3 Warp Map Parameters

This table describes the parameters that specify a user-provided warp map for geometric transformation.

Parameter	Description	Range
numControlPoints	Total number of control points.	uint
warpMapFile	<p>Path to the .map file containing the mapping value of each control point.</p> <p>The size of the file in bytes must be</p> $\text{numControlPoints} * \text{sizeof(float)} * 2.$	string: maximum 1024 characters

6.7.5.7.2.4 TNR Parameters

The following table describes Temporal Noise Reduction parameters.

Parameter	Description	Range
updateTnrParams	Indicates whether to update the TNR parameters for each frame.	uint: [0, 1]
spatialSigmaLuma	Sigma of the luma for spatial filter.	uint
spatialSigmaChroma	Sigma of the chroma for spatial filter.	uint
rangeSigmaLuma	Sigma of the luma for range filter.	uint
rangeSigmaChroma	Sigma of the chroma for range filter.	uint
sadMultiplier	SAD multiplier parameter.	float: [0.0,1.0]
sadWeightLuma	Weight of luma when calculating SAD.	float: [0.0,1.0]
alphaSmoothEnable	Enables or disables the spatial alpha smooth.	uint: [0,1]
alphaIncreaseCap	Temporal alpha restrict increase capability.	float: [0.0,1.0]
alphaScaleIRR	Alpha scale IRR for strength.	float: [0.0,1.0]

Parameter	Description	Range
alphaMaxLuma	Max luma value for alpha clip calculation.	float: [0.0,1.0]
alphaMinLuma	Min luma value for alpha clip calculation.	float: [0.0,1.0]
alphaMaxChroma	Max chroma value for alpha clip calculation.	float: [0.0,1.0]
alphaMinChroma	Min chroma value for alpha clip calculation.	float: [0.0,1.0]
betaX1	BetaX1 in Beta calculation.	float: [0.0,1.0]
betaX2	BetaX2 in Beta calculation.	float: [0.0,1.0]
maxBeta	Max Beta threshold in Beta calculation.	float: [0.0,1.0]
minBeta	Min Beta threshold in Beta calculation.	float: [0.0,1.0]

6.7.5.7.3 Command Line Options

This is the application's command syntax:

```
$ nvmimg_ldc [-h] [-v] [-cf config.conf]
```

Required Options

The following table describes required command line options:

Option	Description	Default
-cf	Specifies the configuration file, including its full path.	(Not Applicable)

Other Options

The following table describes optional command line options:

Option	Description	Default
-h	Displays the application's help text.	(Not Applicable)
-v <level>	Logging level. Value may be: > Errors > Warnings > Info > Debug	0

6.7.5.7.4 Example

This example demonstrates how to call `nvmimg_ldc` to perform an operation.

To copy input into output:

```
$ nvmimg_ldc -cf ldc_basic.conf -v 3
```

Input and output file paths must be edited in the config file.

6.7.5.8 Image 2D (nvmimg_2d)

The NvMedia Image 2D sample application `nvmimg_2d` demonstrates how to use the NvMedia Image 2D API to execute a single NvMedia2DCompose test case defined in .cfg format for NvMediaImage conversion.

The application requires you to provide command line input as well as a group of settings as defined in the configuration file.

Usage

The command syntax is:

```
$ nvmimg_2d <config filename>
```

Configuration File Structure

The configuration file structure consists of parameters, assigned values, and comments. Parameter values are assigned with a line such as `Parameter = Value` in the configuration file. Lines starting with # are comments.

The sample application contains example configuration files (.cfg).

Surface Data File Format

The input and output files (configured with SrcLayerXInputFile and DstOutputFile configuration parameters) contain the surface pixel data as binary dump. The exact layout of the data is determined by the file I/O mode and the surface format set with the configuration parameters.

Configuration Parameters

The following table describes the configuration parameters.

The X in parameters starting with SrcLayerX is an integer in range 1-16.

The Y in parameters containing the string PlaneY is an integer in range 1-3.

Parameter	Description	Type	Range
FileIOMode	<p>Defines how the surface data in input and output files are interpreted.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > • NvSci • LineByLine <p>With NvSci, NvSciBufObjGetPixels() and NvSciBufObjPutPixels() functions are used. This means that for all YUV surface formats, including planar, semiplanar and interleaved, the surface data file format has planar layout with the planes in Y, U, V order.</p> <p>With LineByLine, a simple copy of the surface data occurs line-by-line, discarding any padding.</p> <p>Optional. If not set, defaults to NvSci.</p>	STRING	See description

Parameter	Description	Type	Range
SrcLayerXInputFile	File where the surface data for the source layer is read. Mandatory if the layer is used.	STRING	Max 256 char
SrcLayerXLayout	<p>Memory layout to use for the source surface.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > BlockLinear > PitchLinear <p>Mandatory if the layer is used.</p>	STRING	See description
SrcLayerXScanType	<p>Scan type to use for the source surface.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > Progressive > Interlace <p>Mandatory if the layer is used.</p>	STRING	See description

Parameter	Description	Type	Range
SrcLayerXPlaneYColorFormat	<p>Color format to use for the source surface plane.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > U8V8 > U8_V8 > V8U8 > V8_U8 > U10V10 > V10U10 > U12V12 > V12U12 > U16V16 > V16U16 > Y8 > Y10 > Y12 > Y16 > U8 > V8 > U10 > V10 > U12 > V12 > U16 > V16 > A8Y8U8V8 > Y8U8Y8V8 > Y8V8Y8U8 > U8Y8V8Y8 > V8Y8U8Y8 > A16Y16U16V16 > A8 > B8G8R8A8 > A8R8G8B8 > A8B8G8R8 > A2R10G10B10 > A16B16G16R16 <p>Mandatory if the plane is used.</p>	STRING	See description

Parameter	Description	Type	Range
SrcLayerXPlaneYColorStandard	<p>Color standard to use for the source surface plane.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > SRGB > REC601_SR > REC601_ER > REC709_SR > REC709_ER > REC2020_RGB > REC2020_SR > REC2020_ER > YcCbcCrc_SR > YcCbcCrc_ER > SENSOR_RGBA > REQ2020PQ_ER <p>Optional. If set for one plane, set the value for all planes of the surface.</p>	STRING	See description
SrcLayerXPlaneYWidth	Width of the source surface plane. Mandatory if the plane is used.	UINT	16 - 16384 16 - 8192 if color format with 16-bit components is used.
SrcLayerXPlaneYHeight	Height of the source surface plane. Mandatory if the plane is used.	UINT	16 - 16384 16 - 8192 if color format with 16-bit components is used.
SrcLayerXSrcRectLeft	The source layer's source rectangle left coordinate. Optional. If set, all layer's source rectangle coordinates must be set.	UINT	0 – source surface width

Parameter	Description	Type	Range
SrcLayerXSrcRectTop	The source layer's source rectangle top coordinate. Optional. If set, all layer's source rectangle coordinates must be set.	UINT	0 – source surface height
SrcLayerXSrcRectRight	The source layer's source rectangle right coordinate. Optional. If set, all layer's source rectangle coordinates must be set.	UINT	0 – source surface width
SrcLayerXSrcRectBottom	The source layer's source rectangle bottom coordinate. Optional. If set, all layer's source rectangle coordinates must be set.	UINT	0 – source surface height
SrcLayerXDstRectLeft	The source layer's destination rectangle left coordinate. Optional. If set, all layer's destination rectangle coordinates must be set.	UINT	0 – destination surface width
SrcLayerXDstRectTop	The source layer's destination rectangle top coordinate. Optional. If set, all layer's destination rectangle coordinates must be set.	UINT	0 – destination surface height
SrcLayerXDstRectRight	The source layer's destination rectangle right coordinate. Optional. If set, all layer's destination rectangle coordinates must be set.	UINT	0 – destination surface width

Parameter	Description	Type	Range
SrcLayerXDstRectBottom	The source layer's destination rectangle bottom coordinate. Optional. If set, all layer's destination rectangle coordinates must be set.	UINT	0 – destination surface height
SrcLayerXTransform	<p>Transformation to apply for the source layer.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > None > Rotate90 > Rotate180 > Rotate270 > FlipHorizontal > InvTranspose > FlipVertical > Transpose <p>Optional</p>	STRING	See description
SrcLayerXFiltering	<p>Filtering to apply for the source layer.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > Off > Low > Medium > High <p>Optional</p>	STRING	See description
SrcLayerXBlending	<p>Blending to apply for the source layer.</p> <p>Valid values:</p> <ul style="list-style-type: none"> > Disabled > ConstantAlpha > StraightAlpha > PremultipliedAlpha <p>Optional</p>	STRING	See description

Parameter	Description	Type	Range
SrcLayerXBlendingConsta	Constant alpha factor to use in blending. Mandatory if blending is configured for the layer.	FLOAT	0.0 – 1.0
DstOutputFile	File where the destination surface data is written to. Mandatory.	STRING	Max 256 char
DstLayout	Memory layout to use for the destination surface. See SrcLayerXLayout for valid values. Mandatory.	STRING	See description
DstScanType	Scan type to use for the destination surface. See SrcLayerXScanType for valid values. Mandatory.	STRING	See description
DstPlaneYColorFormat	Color format to use for the destination surface plane. See SrcLayerXPlaneYColorFormat for valid values. Mandatory if the plane is used.	STRING	See description

Parameter	Description	Type	Range
DstPlaneYColorStandard	Color standard to use for the destination surface plane. See <code>SrcLayerXPlaneYColorSt</code> for valid values. Optional. If set for one plane, the value must be set for all planes of the surface.	STRING	See description
DstPlaneYWidth	Width of the destination surface plane. Mandatory if the plane is used.	UINT	16 - 16384 16 - 8192 if color format with 16-bit components is used.
DstPlaneYHeight	Height of the destination surface plane. Mandatory if the plane is used.	UINT	16 - 16384 16 - 8192 if color format with 16-bit components is used.

6.7.5.9 NvMedia IOFA (`nvm_iofa_stereo_sci` and `nvm_iofa_flow_sci`)

Two sample applications, `nvm_iofa_stereo_sci` and `nvm_iofa_flow_sci`, demonstrate how to use the NvMedia Optical Flow Accelerator (OFA) APIs to perform optical flow/stereo matching. The sample uses the NVIDIA OFA hardware engine.

`nvm_iofa_stereo_sci` can read Left View and Right View YUV files and perform stereo matching on input stereo pair. The sample application outputs one stereo disparity value for each $1 \times 1 / 2 \times 2 / 4 \times 4 / 8 \times 8$ input block on the output surface. Grid size controls input block size, which is sample application's argument.

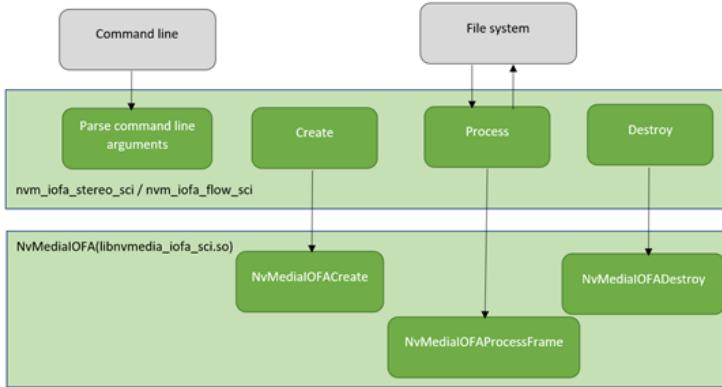
The output stereo disparity format is fixed signed 10.5 and range of disparity is fixed [0, 127] or [0, 255]. Divide the output values by 32 to get a disparity value in terms of pixel units.

`nvm_iofa_flow_sci` can read input YUV file and perform optical flow estimation on input YUV file. The sample outputs one flow vector value for each $1 \times 1 / 2 \times 2 / 4 \times 4 / 8 \times 8$ input block on the output surface. Input block size is controlled by grid size, which is sample application's argument.

The output flow vector format is fixed signed 10.5 [Range -1024 to 1023]. For optical flow, divide each component of flow vector values by 32 to get the flow vector in terms of pixel units.

Architecture

The following figure shows the architecture of the `nvm_iofa_stereo_sci` and `nvm_iofa_flow_sci` application.



To run the sample:

1. Follow the steps in [Building and Running the NvMedia Samples](#).
2. Launch the application:

```
$ nvm_iofa_stereo_sci -h -v 3
```

```
$ nvm_iofa_flow_sci -h -v 3
```

Command Line Options for `nvm_iofa_stereo_sci`

The sample application command syntax is as follows:

```
$ nvm_iofa_stereo_sci [options]
```

The command line options are as follows.

String parameters, such as pathnames, have a maximum length of 1024 characters. Restrictions on other types of parameters are noted individually.

Required Command Line Options

Option	Parameter	Description
-l	<i>Left view file name</i>	Specifies left view file name with full path.
-r	<i>Right view file name</i>	Specifies right view file name with full path.
-res	<w>x<h>	<p>Specifies the input surface dimensions, where <w> represents width and <h> represents height. Width and height are separated by a letter 'x' with no spaces, for example:</p> <p style="margin-left: 40px;">-res 640x480</p> <p><w> has a range of [32, 8192].</p> <p><h> has a range of [32, 8192].</p>

Optional Command Line Options

Option	Parameter	Description
-h	n/a	Displays guidance on using this application.
-version	n/a	Get NvMediaOFA Major/Minor version
-v	0: Errors (default) 1: Warnings 2: Info 3: Debug	Logging level.
-gridSize	0: Grid Size 1x1 1: Grid Size 2x2 2: Grid Size 4x4 3: Grid Size 8x8	Output Grid Size set by Application.
-o	Disparity Output file name	Specifies disparity output file name with full path.

Optional Command Line Options

Option	Parameter	Description
-co	Cost Output file name	Specifies cost output file name with full path.
-stereocrcgen	A pathname of a file to be created with the filetype .txt	Generates CRC values of disparity map in the specified file.
-stereocrchk	A pathname of an existing file with the filetype .txt	Checks CRC values generated from disparity map against values from the specified file.
-costcrcgen	A pathname of a file to be created with the filetype .txt	Generates CRC values of cost surface in the specified file.
-costcrchk	A pathname of an existing file with the filetype .txt	Checks CRC values generated from cost surface against values from the specified file.
-if	0: YV12 (default) 1: IYUV	Input YUV file format.
-chromaFormat	0 : 400 1 : 420 2 : 422 3 : 444	Chroma Format IDC of input yuv
-ndisp	128 256	Max Disparity value in terms of pixels
-rlSearch	0: Disable reverse search (default) 1: Enable reverse search	Enable reverse search generates right view disparity map.
-frames	An integer	Specifies the number of input frames to be processed
-inputBuffering	1 to 8 default (1)	Input YUV pair buffering

Optional Command Line Options

Option	Parameter	Description
-profile	0: Disable Profile 1: Enable Profile with Async mode 2: Enable Profile with Sync mode	Enable OFA profiling to get SW and HW overhead
-etype	0: Stereo Disparity HQ mode (default)	Specifies stereo matching quality mode.
-bit_depth	8: 8 bits (default) 10: 10 bits 12: 12 bits 16: 16 bits	Number of bits per components on input surfaces.
-timeout	An Integer (default – 30ms)	timeout value to wait for OFA operation to finish in ms
-p1	An integer	SGM penalty1 value
-p2	An integer	SGM penalty2 value
-diag	0: Disable diagonal mode 1: Enable diagonal mode	SGM diagonal mode
-adaptiveP2	0: Disable adaptive P2 1: Enable adaptive P2	SGM adaptive p2 mode enabled
-alpha	0 to 3	alpha value to use along with adaptiveP2
-pass	1 to 3	Num Passes of SGM

Optional Command Line Options

Option	Parameter	Description
-frameIntervalInMS	An Integer	Time interval between two stereo pair in milisec.
-roiMode	0: Disable ROI mode (default) 1: Enable ROI mode	ROI (Region of interest) mode
-roiPosXStartDiv32 -roiPosYStartDiv8 -roiWidthDiv32 -roiHeightDiv8	An Integer	ROI parameters ROI top-left x position (in 32-pixel unit) ROI top-left y position (in 8-pixel unit) ROI width (in 32-pixel unit) ROI height (in 8-pixel unit)

Examples

These examples demonstrate how to call `nvm_iofa_stereo_sci` to perform various operations.

> **Stereo Disparity estimations for YUV subsampling type 420 8-bit input**

```
$ nvm_iofa_stereo_sci -l left_inputfile -r rightinputfile -res wxh -o outputfile -  
inputBuffering 1 -frames 100 -v 3
```

The application performs stereo matching according to the parameters in the command line.

> **Stereo Disparity estimation for YUV subsampling type 400 10-bit input**

```
$ nvm_iofa_stereo_sci -l leftinputfile -r rightinputfile -res wxh -o outputfile -  
inputBuffering 3 -chromaFormat 0 -bit_depth 10 -frames 100 -v 3
```

The application performs stereo matching according to the parameters in the command line.

> **Stereo Disparity estimation for YUV subsampling type 444 16-bit input**

```
$ nvm_iofa_stereo_sci -l leftinputfile -r rightinputfile -res wxh -o outputfile -  
inputBuffering 2 -chromaFormat 3 -bit_depth 16 -frames 100 -v 3
```

The application performs stereo matching according to the parameters in the command line.

- > **Stereo Disparity estimation for YUV subsampling type 420 8-bit input with CRC generation**

```
$ nvm_iofa_stereo_sci -l leftinputfile -r rightinputfile -res wxh -o outputfile -  
inputBuffering 2 -stereocrcgen crc.txt -frames 100 -v 3
```

The application performs stereo matching according to the parameters in the command line.

- > **Stereo Disparity estimation for YUV subsampling type 420 8-bit input with CRC comparison**

```
$ nvm_iofa_stereo_sci -l leftinputfile -r rightinputfile -res wxh -o outputfile -  
inputBuffering 2 -stereocrcchk crc.txt -frames 100 -v 3
```

The application performs stereo matching according to the parameters in the command line.

Command Line Options for nvmedia_ofa_flow

The sample application command syntax is as follows:

```
$ nvm_iofa_flow_sci [options]
```

The command line options are as follows.

String parameters, such as pathnames, have a maximum length of 1024 characters. Restrictions on other types of parameters are noted individually.

Required Command Line Options		
Option	Parameter	Description
-f	<i>Input file name</i>	Specifies input file name with full path.
-res	<w>x<h>	Specifies the input surface dimensions, where <w> represents width and <h> represents height. Width and height are separated by a letter 'x' with no spaces, for example: -res 640x480 <w> has a range of [32, 8192]. <h> has a range of [32, 8192].

Optional Command Line Options

Option	Parameter	Description
-h	n/a	Displays guidance on using this application.
-version	n/a	Get NvMediaOFA Major/Minor version
-v	0: Errors (default) 1: Warnings 2: Info 3: Debug	Logging level.
-gridSize	0: Grid Size 1x1 1: Grid Size 2x2 2: Grid Size 4x4 3: Grid Size 8x8	Output Grid Size set by Application.
-o	Flow Output file name	Specifies flow output file name with full path.
-co	Cost Output file name	Specifies cost output file name with full path.
-flowcrcgen	<i>A pathname of a file to be created with the filetype .txt</i>	Generates CRC values of flow map in the specified file.
-flowcrcchk	<i>A pathname of an existing file with the filetype .txt</i>	Checks CRC values generated from flow map against values from the specified file.
-costcrcgen	<i>A pathname of a file to be created with the filetype .txt</i>	Generates CRC values of cost surface in the specified file.
-costcrcchk	<i>A pathname of an existing file with the filetype .txt</i>	Checks CRC values generated from cost surface against values from the specified file.
-if	0: YV12 (default) 1: IYUV	Input YUV file format.

Optional Command Line Options

Option	Parameter	Description
-chromaFormat	0 : 400 1 : 420 (default) 2 : 422 3 : 444	Chroma Format IDC of input yuv (level 0 / base layer for flow estimation).
-pydChromaFormat	0 : 400 1 : 420 (default) 2 : 422 3 : 444	Chroma Format IDC of input pyramid except base layer.
-frames	An integer	Specifies the number of input frames to be processed
-inputBuffering	1 to 8 <i>default (1)</i>	Input YUV pair buffering
-profile	0: Disable Profile 1: Enable Profile with Async mode 2: Enable Profile with Sync mode	Enable OFA profiling to get SW and HW overhead
-etype	0: Optical Flow HQ mode (default)	Specifies of matching quality mode.
-bit_depth	8: 8 bits (default) 10: 10 bits 12: 12 bits 16: 16 bits	Number of bits per components on input surfaces.
-timeout	An Integer (default – 30ms)	timeout value to wait for OFA operation to finish in ms
-p1	An integer	SGM penalty1 value
-p2	An integer	SGM penalty2 value

Optional Command Line Options

Option	Parameter	Description
-diag	0: Disable diagonal mode 1: Enable diagonal mode	SGM diagonal mode
-adaptiveP2	0: Disable adaptive P2 1: Enable adaptive P2	SGM adaptive p2 mode enabled
-alpha	0 to 3	alpha value to use along with adaptiveP2
-pass	1 to 3	Num Passes of SGM
-frameIntervalInMS	An Integer	Time interval between two stereo pair in milisec.
-roiMode	0: Disable ROI mode (default) 1: Enable ROI mode	ROI (Region of interest) mode
-roiPosXStartDiv32 -roiPosYStartDiv8 -roiWidthDiv32 -roiHeightDiv8	An Integer	ROI parameters ROI top-left x position (in 32-pixel unit) ROI top-left y position (in 8-pixel unit) ROI width (in 32-pixel unit) ROI height (in 8-pixel unit)

Examples

These examples demonstrate how to call `nvm_ifoa_flow_sci` to perform various operations.

> **Flow estimation for YUV subsampling type 420 8-bit input**

```
$ nvm_ifoa_flow_sci -f inputfile -res wkh -o outputfile -inputBuffering 1 -frames 100
-v 3
```

- > The application performs flow estimation according to the parameters in the command line.

> **Flow estimation for YUV subsampling type 400 10-bit input**

```
$ nvm_ifoa_flow_sci -f inputfile -res wkh -o outputfile -inputBuffering 3 -
chromaFormat 0 -bit_depth 10 -frames 100 -v 3
```

The application performs flow estimation according to the parameters in the command line.

> **Flow estimation for YUV subsampling type 444 16-bit input**

```
$ nvm_iofa_flow_sci -f inputfile -res wxh -o outputfile -inputBuffering 2 -chromaFormat 3 -bit_depth 16 -frames 100 -v 3
```

The application performs flow estimation according to the parameters in the command line.

> **Flow estimation for YUV subsampling type 420 8-bit input with CRC generation**

```
$ nvm_iofa_flow_sci -f inputfile -res wxh -o outputfile -inputBuffering 2 -flowcrcgen crc.txt -frames 100 -v 3
```

The application performs flow estimation according to the parameters in the command line.

> **Flow estimation for YCbCR subsampling type 420 8-bit input with CRC comparison**

```
$ nvm_iofa_flow_sci -f inputfile -res wxh -o outputfile -inputBuffering 2 -flowcrcgen crc.txt -frames 100 -v 3
```

The application performs flow estimation according to the parameters in the command line.

6.7.5.10 Optisense (stereosense and flowsense)

Two sample applications, stereosense and flowsense, are part of Optisense apps. These applications demonstrate an end-to-end pipeline implementation of OFA use cases, which involve pre-processing and post-processing modules along with OFA HW processing. The post-processing and pre-processing modules of these sample applications are implemented on CPU, which can be later ported on hardware modules such as 2D and CUDA.

stereosense – The test application demonstrates stereo pipeline. stereosense reads the Left View, Right View YUV files, and performs stereo matching on input stereo pairs. The disparity output is post-processed with median filtering, nearest neighbor up-sampling, and left-right consistency checks.

The output stereo disparity format is fixed signed 10.5 and range of disparity is fixed to [0, 127] or [0, 255]. Divide the output values by 32 to get a disparity value in pixels.

flowsense – The test application demonstrates the optical flow pipeline. flowsense reads the input YUV file and performs optical flow estimation on the input YUV file. flowsense generates flow map for input frames provided and post-processes the flow output with median filtering and nearest neighbor up-sampling.

Application generates an image pyramid of input and reference frames, which is provided as input to OFA HW.

6.7.5.10.1 Running the Sample Application

To run the sample:

1. Follow the steps in [Building and Running the NvMedia Samples](#).
2. Launch the application:

```
$ stereosens -h
```

```
$ flowsense -h
```

The sample application command syntax is as follows:

for stereo:

```
$ stereosense [options]
```

The command line options are as follows:

Required Command Line Options		
Option	Parameter	Description
-input	Left view file name	Specifies left view file name with full path.
-ref	Right view file name	Specifies right view file name with full path.
-width	width of input	Specifies the width of input surface
-height	height of input	Specifies the height of input surface

Optional Command Line Options		
Option	Parameter	Description
-h	n/a	Displays guidance on using this application.
-gridSize	0: Grid Size 1x1 1: Grid Size 2x2 2: Grid Size 4x4 3: Grid Size 8x8	Output Grid Size set by Application.
-o	Disparity Output file name	Specifies disparity output file name with full path.
-co	Cost Output file name	Specifies cost output file name with full path.

Required Command Line Options

Option	Parameter	Description
-flowcrcgen	A pathname of a file to be created with the filetype .txt	Generates CRC values of disparity map in the specified file.
-flowcrcchk	A pathname of an existing file with the filetype .txt	Checks CRC values generated from disparity map against values from the specified file.
-chromaFormat	0 : 400 1 : 420 2 : 422 3 : 444	Chroma Format IDC of input yuv
-ndisp	128 256	Max Disparity value in terms of pixels
-lrCheck	0: Disable reverse search (default) 1: Enable reverse search	Enable reverse search generates right view disparity map.
-lrCheckThr	An integer	Decides threshold for lrcheck
-nframes	An integer	Number of frames used for stereo estimation. Default: one frames
-profile	0: Disable Profile 1: Enable Profile with Async mode : Enable Profile with Sync mode	Enable OFA profiling to get SW and HW overhead
-etype	0: use default SGM parameters value (default) 1: use configurable Values for SGM parameters	Enable configurable SGM parameters

Required Command Line Options		
Option	Parameter	Description
-bitDepth	8: 8 bits (default) 10: 10 bits 12: 12 bits 16: 16 bits	Number of bits per components on input surfaces.
-p1	An integer	SGM penalty1 value
-p2	An integer	SGM penalty2 value
-diag	0: Disable diagonal mode 1: Enable diagonal mode	SGM diagonal mode
-adaptiveP2	0: Disable adaptive P2 1: Enable adaptive P2	SGM adaptive p2 mode enabled
-alpha	0 to 3	alpha value to use along with adaptiveP2
-pass	1 to 3	Num Passes of SGM
-median	0: Disable median filtering (default) 1: Enable median filtering	Enable median filtering over output
-upsample	0: Disable upsampling (default) 1: Enable upsampling	Enable upsampling of output

6.7.5.10.2 Example: flowsense and Post Median Filtering

To run flowsense without post median filtering

```
./flowsense -input left_000165_1238x374.yuv -width 1238 -height 374 -  
bitDepth 8 -gridSize 2 -chromaFormat 0 -o flow.bin -co cost.bin -flowcrcgen  
flow.txt -nframes 20
```

To run flowsense with median filtering

```
./flowsense -input left_000165_1238x374.yuv -width 1238 -height 374 -  
bitDepth 8 -gridSize 2 -chromaFormat 0 -o flow.bin -co cost.bin -flowcrcgen  
flow.txt -median 1 -nframes 2
```

6.7.5.10.3 Example: stereosense and Median Filtering

To run stereosense without median filtering:

```
./stereosense -input left_000165_1238x374.yuv -ref right_000165_1238x374.yuv
-width 1238 -height 374 -flowcrcgen stereo.txt -nframes 20 -gridSize 3 -
bitDepth 8 -chromaFormat 0 -ndisp 256 -o stereo.bin
```

To run stereosense with median filtering:

```
./stereosense -input left_000165_1238x374.yuv -ref right_000165_1238x374.yuv
-width 1238 -height 374 -flowcrcgen stereo.txt -nframes 20 -gridSize 3 -
bitDepth 8 -chromaFormat 0 -ndisp 256 -median 1 -o stereo.bin
```

for flow:

\$ flowsense [options]

The command line options are as follows.

Required Command Line Options		
Option	Parameter	Description
-input	<i>Left view file name</i>	Specifies input file name with full path.
-ref	<i>Right view file name</i>	Specifies reference file name with full path.
-width	width of input	Specifies the width of input surface
-height	height of input	Specifies the height of input surface

Optional Command Line Options		
Option	Parameter	Description
-h	n/a	Displays guidance on using this application.
-gridSize	0: Grid Size 1x1 1: Grid Size 2x2 2: Grid Size 4x4 3: Grid Size 8x8	Output Grid Size set by Application.
-o	Disparity Output file name	Specifies disparity output file name with full path.

Required Command Line Options

Option	Parameter	Description
-co	Cost Output file name	Specifies cost output file name with full path.
-flowcrcgen	<i>A pathname of a file to be created with the filetype .txt</i>	Generates CRC values of disparity map in the specified file.
-flowcrcchk	<i>A pathname of an existing file with the filetype .txt</i>	Checks CRC values generated from disparity map against values from the specified file.
-chromaFormat	0 : 400 1 : 420 2 : 422 3 : 444	Chroma Format IDC of input yuv
-do_bwd	Enable backward reference	Use backward referencing for flow
-nframes	<i>An integer</i>	Number of frames used for flow estimation. Default: one pair frames
-profile	<i>0: Disable Profile</i> <i>1: Enable Profile with Async mode</i> <i>2: Enable Profile with Sync mode</i>	Enable OFA profiling to get SW and HW overhead
-bitDepth	8: 8 bits (default) 10: 10 bits 12: 12 bits 16: 16 bits	Number of bits per components on input surfaces.
-p1	An integer	SGM penalty1 value
-p2	An integer	SGM penalty2 value
-diag	<i>0: Disable diagonal mode</i> <i>1: Enable diagonal mode</i>	SGM diagonal mode

Required Command Line Options		
Option	Parameter	Description
-adaptiveP2	0: Disable adaptive P2 1: Enable adaptive P2	SGM adaptive p2 mode enabled
-alpha	0 to 3	alpha value to use along with adaptiveP2
-pass	1 to 3	Num Passes of SGM
-median	0: Disable median filtering (default) 1:Enable median filtering	Enable median filtering over output
-upsample	0: Disable upsampling (default) 1:Enable upsampling	Enable upsampling of output

6.7.5.11 Deep Learning Accelerator Programming Interface (nvm_dlaSample)

The NvMedia nvm_dlaSample sample application demonstrates how to use the NvMedia Deep Learning Accelerator (DLA) APIs to perform deep learning inference operations. The sample uses the NVIDIA® SoC DLA hardware engine.

The nvm_dlaSample application has four testing modes:

1. Runtime mode.
 - > Single thread for running DLA.
 - > Demonstrates how to create and initialize DLA instances.
 - > Demonstrates how to run DLA with provided input data and network.
2. SciSync mode.
 - > Single thread for running DLA. Two supporting threads for synchronization (signaler and waiter).
 - > Demonstrates how to create and initialize DLA instances.
 - > Demonstrates how to run DLA with provided input data and network.
 - > Demonstrates how to synchronize DLA task submission with a CPU signaler and CPU waiter.
3. Multithreaded mode.
 - > Multiple threads (4) for running DLA.
 - > Demonstrates how to create and initialize DLA instances.
 - > Demonstrates how to run DLA with provided input data and network.

4. Ping mode.

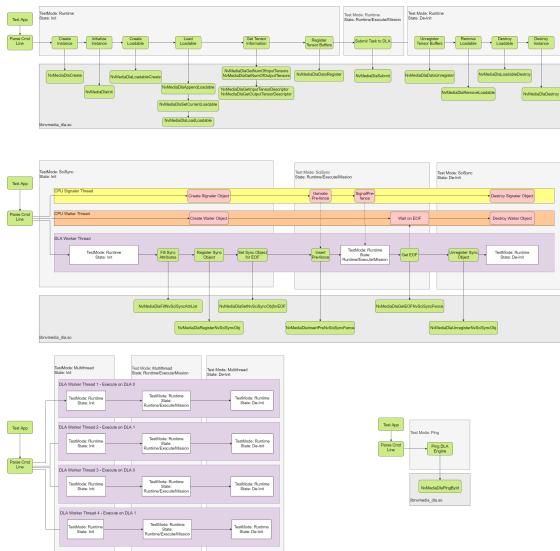
- Pings the specified instance if it exists.



Note: The tegrastats utility enables and reports on resource utilization. For additional information, refer to [DLA in GitHub](#).

6.7.5.11.1 Architecture

The following diagram shows the architecture of the nvm_dlaSample application.



Click the image to enlarge it.

6.7.5.11.2 Running the Sample Application

6.7.5.11.2.1 Prerequisites

You have followed the procedures in [Building and Running the NvMedia Samples in NvMedia Sample Applications](#).

6.7.5.11.2.2 To run the sample

1. Navigate to:

```
samples/nvmedia/dla_sample/
```

2. Launch the application:

```
$ nvm_dlaSample --mode runtime -d instanceId -n numTasks --loadable /path/to/loadable
```

6.7.5.11.3 Command Line Switches

This is the nvm_dlaSample command syntax:

```
$ nvm_dlaSample [-h] [-v <Logging Level>] [--mode <Mode>] [-d <Instance ID>] [-n numTasks] [--loadable /path/to/loadable]
```

The following table describes the command line switches.

Switch	Para-meter	Description
-h	N/A	Displays guidance on using this application.
-d	[Instance ID]	Specifies the hardware engine ID on which to run the test. The value may be 0 or 1. The default is 0.
-n	[numTasks]	Specifies the number of simultaneous tasks that can be sent to a DLA instance.
--mode	[mode]	Specifies the mode of the test to run. The value may be: > runtime > scisync > multithread > ping
--loadable	[path]	Path to the loadable file.
-v	N/A	Enables verbose descriptions.

6.7.5.11.4 NvMedia Sample Applications

The NvMedia sample applications demonstrate how to use the NvMedia API to perform multimedia tasks. Multimedia based samples are built for single windowing compatibility that depends on the platform operating system.

NVIDIA DRIVE 6.0 Linux uses the X11 windowing system.

Graphics OpenGL ES samples are built and organized for compatibility with both Wayland and X11 windowing systems with subdirectories for both. For information on the graphics OpenGL ES samples, see [Building and Running Samples in Graphics Programming](#).

6.7.5.11.4.1 Runtime mode

```
$ nvm_dlaSample --mode runtime -d 0 -n 8 --loadable /path/to/loadable
```

6.7.5.11.4.2 Scisync mode

```
$ nvm_dlaSample --mode scisync -d 0 -n 8 --loadable /path/to/loadable
```

6.7.5.11.4.3 Multithread mode

```
$ nvm_dlaSample --mode multithread --loadable /path/to/loadable
```

6.7.5.11.4.4 Ping mode

```
$ nvm_dlaSample --mode ping
```

6.7.6 Debugging CSI Capture Errors



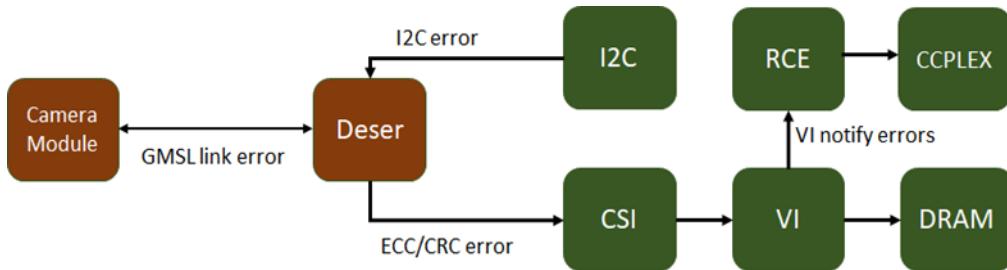
Note:

All commands in this section must run as root user (with sudo).

The terms used throughout this section are as follows:

- > **CCPLEX**: NVIDIA Carmel CPU complex running the capture stack and applications.
- > **CRC**: Cyclic Redundancy Check, a 16-bit checksum present in MIPI CSI packet footer.
- > **ECC**: Error Correction Code, a 6-bit code present in MIPI CSI packet header that can be used by the receiver to correct single bit errors and detect double bit errors.
- > **NVCSI**: An SoC MIPI CSI interface conformant to CSI2 v2.0 specification.
- > **RCE**: Realtime Camera Engine is an ARM Cortex R5 core running an RTOS that handles programming of NVCSI/VI engines and notify messages from VI.
- > **VI**: An SoC Video Input module that takes frame data from NVCSI interface and writes it to memory.

The layers where errors can occur during a camera capture are as follows:



Deserializer Errors

Two signals that can be generated by Deserializer are as follows:

1. ERRB
2. LOCK

If these signals are toggled, SIPL sends error notification to the client, and the client collects the detailed error information from the hardware.

6.7.6.1 Error Logs

When an error is encountered in decoding CSI packets received at an SoC CSI interface and writing the raw frame data to memory, the VI hardware engine notifies these errors to RCE. The capture stack running on CCPLEX queries the capture status from RCE and displays:

- > The CSI stream ID (CSI brick).
- > VC ID for which the error occurred.
- > The type of error.
- > Detailed errors per the error type.

The error status provides a good starting point to determine the root cause and identify the next steps.

If the VI engine did not successfully capture a frame, and did not encounter and report an error to RCE, then the error message, as follows, or a similar one displays:

Frame Start Timeout Error.

```
EngineStatusError (2), Detailed Error Code: 00000001 at
StreamId:0 vcId:0 CSIFrameId:0 notified-bits:2000000or
```

Frame End Timeout Error

```
EngineStatusError (2), Detailed Error Code: 00000002 at
StreamId:0 vcId:0 CSIFrameId:0 notified-bits:4000000
```

This can be due to one of two reasons:

1. The deserializer is not streaming data.
2. The VI channel is not configured to capture the correct data type/VC id.

The remainder of this document describes the root causes for these errors.

6.7.6.2 RCE Traces

Apart from the error events, the VI notifies several other events to RCE that can be read by CCPLEX using the RCE tracing framework. This notification is disabled by default and can be enabled as follows:

1. Run these commands to enable the RCE traces before starting capture:

```
echo 10000 > /sys/kernel/debug/tracing/buffer_size_kb
echo 1 > /sys/kernel/debug/tracing/tracing_on
echo 1 > /sys/kernel/debug/tracing/events/tegra_rtcpu/enable
echo 3 > /sys/kernel/debug/camrtc/log-level
```

2. After starting capture, read the RCE traces using:

```
cat /sys/kernel/debug/tracing/trace
```

3. Clear the RCE trace from the previous capture using:

```
echo > /sys/kernel/debug/tracing/trace
```

The RCE trace logs several events such as SOF, EOF that can be used to determine if the VI channel is capturing data or not. Capture the RCE traces and provide them to NVIDIA for further analysis.

6.8 Programmable Vision Accelerator (PVA)

NVIDIA DRIVE® OS provides support for the Programmable Vision Accelerator (PVA) engine. The PVA engine is an advanced VLIW SIMD digital signal processor, which is optimized to implement image processing and computer vision algorithms. PVA

provides excellent performance with extremely low power consumption. PVA can be used asynchronously and concurrently with the CPU, GPU, and other accelerators on the DRIVE platform as part of a heterogeneous compute pipeline.

If you are planning to build PVA applications or to use PVA applications built in to DriveWorks, update the PVA allowlist located at the following IFS path:

```
<rootfs>/lib/firmware/pva_auth_allowlist
```

NVIDIA Driveworks is the primary API for the PVA engine. Refer to Driveworks API documentation for supported algorithms and usage.

With additional software, it is also possible to program the PVA directly with custom algorithms. Contact NVIDIA for more information.

6.9 Logging



Note: The nvlog feature is currently under development. You can enable this feature for testing purposes, but do not enable the feature in production systems until it is enabled by default in NVIDIA DRIVE® OS releases.

NVIDIA DRIVE® OS has a logging component that allows logs from multiple NVIDIA DRIVE OS components to be obtained. Currently, support is only available for applications within NVIDIA DRIVE OS Linux Guest VM to send logs that are stored in a file that is created and accessible from Linux Guest VM. This support will be enhanced in future releases to support more execution environments outside of Guest VM.

Terms

Logging Entity : Application threads(s) trying to log, each thread is an entity.

Entity Logging library : entity specific logging library build that is linked to Guest VM entities (`libnvlog_guestvm_entity`)

`nvlogclient_fast` : logging client/server combined application.

Logging library : full build of logging library that is linked to `nvlogclient_fast` (`libnvlog.so`)

Logging Pipeline

The logging pipeline is as follows:

```
(Logging Entity + libnvlog_guestvm_entity) -> (nvlogclient_fast + libnvlog.so) -> log storage file
```

Components

The following lists the source and binary locations within the SDK:

1. Pre-built binaries:

```
<top>/drive-linux/lib-target/libnvlog.so
```

```
<top>/drive-linux/filesystem/contents/bin/nvlogclient_fast
<top>/drive-linux/filesystem/contents/bin/nvlogtest
<top>/drive-linux/filesystem/contents/bin/nvlogtest_stdin
```

2. Source code:

Source files for the test applications (nvlogtest*) and the Logging Client (nvlogclient_fast) can be found under:

```
<top>/drive-linux/samples/tools/nvlog/
```

3. Include files and user API:

Any application needing to send logs using this new framework must use the `<top>/drive-linux/samples/tools/nvlog/include/nvlog.h` header file and link to `libnvlog_guestvm_entity` library. API documentation can be found in the same header file.

In a booted DRIVE OS Linux Guest VM console, the following logging-related components are available:

1. /usr/local/bin/

`nvlogclient_fast`: Application component responsible for collecting logs from different Logging Entities within Linux Guest VM and storing them.

`nvlogtest`: Test application that has two threads sending log messages every second.

`nvlogtest_stdin`: Test application that takes input from stdin and sends each line as a log message.

2. /usr/local/lib/

`libnvlog.so`: Logging library that provides logging APIs.

Execution Steps

Default Behavior

The nvlog feature is enabled by default on AV + L for all the platforms. The following messages appear during boot:

```
Dec 21 00:54:11 tegra-ubuntu systemd[1]: Starting nvlog client logging service...
Dec 21 00:54:11 tegra-ubuntu bash[1140]: Launched /usr/local/bin/nvlogclient_fast !!!
Dec 21 00:54:11 tegra-ubuntu nvlogclient_fast: SHMDiscoveryClientThread: Discovery
client thread starting...
Dec 21 00:54:11 tegra-ubuntu nvlogclient_fast: NvLogDiscoveryPlugin_nvsciipc_Init -
Entry
Dec 21 00:54:11 tegra-ubuntu systemd[1]: Started nvlog client logging service.
```

For the standard build, the output log file is created in the `/home/nvidia/` directory:

```
nvidia@tegra-ubuntu:~$ ls -al /home/nvidia/nvlog_system_log.txt
-rw-r--r--. 1 root root 20480 Dec 21 00:54 /home/nvidia/nvlog_system_log.txt
```

```
nvidia@tegra-ubuntu:~$ cat /home/nvidia/nvlog_system_log.txt
EUID      PID ENTITY ID    SEQ      TIMESTAMP   MSG
```

```

          0      1305      1305            3      346841060  Module_id 51 Severity
5 : created TA endpoint: 1, created CA endpoints: 2
          0      1305      1305            2      346840076  Module_id 51 Severity
5 : Server trying IVC endpoint: pkcs11_keystore_tk11_ivc_d2
          0      1305      1305            1      346838406  Module_id 51 Severity
5 : Server trying IVC endpoint: pkcs11_keystore_tk11_ivc_d1
          0      1291      1304            1      346765176  Module_id 51 Severity
5 : Waiting for Suspend Request from Kernel: pkcs11-keystore-tk10
          0      1291      1291            3      346758455  Module_id 51 Severity
5 : created TA endpoint: 1, created CA endpoints: 2
          0      1291      1291            2      346757262  Module_id 51 Severity
5 : Server trying IVC endpoint: pkcs11_keystore_tk10_ivc_d2
          0      1291      1291            1      346755441  Module_id 51 Severity
5 : Server trying IVC endpoint: pkcs11_keystore_tk10_ivc_d1
          0      1284      1299            1      346717953  Module_id 51 Severity
5 : Waiting for Suspend Request from Kernel: pkcs11-keystore-tk1
          0      1284      1284            3      346711198  Module_id 51 Severity
5 : created TA endpoint: 1, created CA endpoints: 2
          0      1284      1284            2      346710136  Module_id 51 Severity
5 : Server trying IVC endpoint: pkcs11_keystore_tk1_ivc_d2
          0      1284      1284            1      346708419  Module_id 51 Severity
5 : Server trying IVC endpoint: pkcs11_keystore_tk1_ivc_d1
          0      1272      1281            1      346543194  Module_id 51 Severity
5 : Waiting for Suspend Request from Kernel: nvmacsec
          0      1272      1272            3      346537165  Module_id 51 Severity
5 : created TA endpoint: 1, created CA endpoints: 2
          0      1272      1272            2      346536002  Module_id 51 Severity
5 : Server trying IVC endpoint: nvmacsec_ta_d2
          0      1272      1272            1      346534344  Module_id 51 Severity
5 : Server trying IVC endpoint: nvmacsec_ta_d1
          0      1260      1270            1      346428824  Module_id 51 Severity
5 : Waiting for Suspend Request from Kernel: gp-se
...

```

For the production build, the output log file is created in the /var directory.

Disable nvlog

To disable nvlog,

1. Change the status of the nvlog node to disabled in the tegra234-linux-gos.dtsi file.

```

nvlog {
    status = "disabled";
    guest_vm {
        ...
    }
}

```

2. Recompile and flash.



Note:

- > Raw output from Logging Client is not sequential with regard to generated logs. This is due to the underlying implementation where readers and writers are lockless and the reader retrieves messages in bursts (in reverse order for each burst). The raw output file must be sorted on the basis of timestamp (column 5) to get the logs from multiple sources in order (example: sort -k5 -n).
- > Log messages can be lost if entity (or producer) is spewing many log messages before nvlogclient_fast (or consumer) is able to read the messages.
- > The first log message is not currently retrieved or stored by the Logging Client. This is a known issue.

6.10 Docker Services

NVIDIA DRIVE® OS provides a Docker containerization service as part of the NVIDIA DRIVE® OS Linux Guest VM. The service integrates both the Docker runtime environment as well as the NVIDIA Container Toolkit to provide containerization with iGPU access on the target. For information on how to run CUDA samples inside target-side Docker container and how to run custom applications inside a target-side Docker container, refer to the [Running Docker Containers Directly on NVIDIA DRIVE AGX Orin](#) technical blog.

Docker services contain the following components:

- > [Docker Runtime](#)
- > [NVIDIA Container Toolkit](#)
- > [Sample: How to Launch](#)
- > [Support for Other Applications](#)

Docker Runtime

Docker is a high-level API that complements kernel namespaces to provide a Linux container runtime with strong guarantees of isolation and repeatability. The runtime provides both a daemon, which implements the containerization, and a client to interact with said daemon.

The Docker client can be run by executing `sudo docker` on the target. You can confirm that the daemon is running by executing `sudo service docker status`.

The Docker runtime present in the NVIDIA DRIVE OS Linux Guest VM is from upstream Ubuntu Canonical repositories and is considered experimental on non-amd64 architectures.

To learn more in depth about Docker, the runtime, and features, visit <https://docker.com>.

NVIDIA Container Toolkit

The NVIDIA Container Runtime uses this specification to determine which directories, devices, libraries, and files to make available to a Docker container at runtime. Through the Mount Plugins Specification, it is possible to run GPU-accelerated applications on the target within a Docker container that would otherwise have been isolated from the iGPU.

> /etc/nvidia-container-runtime/host-files-for-container.d/devices.csv

Specifies the list devices for the Mount Plugins Specification.

> /etc/nvidia-container-runtime/host-files-for-container.d/drivers.csv

Specifies the libraries for the Mount Plugins Specification

The preceding two files are provided as part of the NVIDIA DRIVE OS Linux Guest VM file system to enable out-of-the-box support for running CUDA samples on the target within Docker. They provide instructions for the Mount Plugins Specification for mounting the required directories, devices, libraries, and so on, into Docker.

The section below goes through a simple step-by-step guide on how to use Docker on the target and use the NVIDIA Container Runtime to run a GPU-accelerated CUDA sample from within Docker.

Sample: How to Launch

The following sample uses the deviceQuery application that is provided as part of the CUDA installation on the target. The following also assumes that the platform is connected to the Internet. Internet access will be required for step 3 as the Docker runtime will attempt to pull or download the ubuntu:20.04 image from Docker's repository.

1. Change directory to the path for the deviceQuery sample and compile it.

```
cd /usr/local/cuda-11.4/samples/1_Utilities/deviceQuery/ && sudo make
```

2. Run the deviceQuery sample and confirm successful output for GPU device information:

```
./deviceQuery
```

3. Execute Docker with the given sample to confirm iGPU access from within Docker. It should print to console the same output that was printed in step 2.

```
sudo docker run --rm --runtime nvidia --gpus all -v $(pwd):$(pwd) -w $(pwd) ubuntu:20.04 ./deviceQuery
```

This command might take a few moments as it will need to pull the Ubuntu 20.04 Docker image and start a Docker terminal session.

Support for Other Applications

The devices.csv and drivers.csv files are configured out of the box to support successfully running the deviceQuery CUDA sample application within Docker. Supporting other GPU-accelerated applications only requires adding the appropriate

paths to dependent libraries, devices, directories, and so on, to the `devices.csv` and `drivers.csv` files.

To run through an example workflow, use the NVIDIA DriveWorks `hello_world` sample application.

6.10.1 Supporting the sample_hello_world Application

Application developers often deal with nested dependencies. Sometimes these dependencies might have requirements that are impossible to identify even from reading a vendor's developer guide. For example, a dependency two or more layers down requires the ability to write to a location on the host system. The dependency assumes that the location exists on the host, but it actually does not. At which point, the developer encounters an error.



Note:

- > This topic is intended to show ways to identify dependencies, debug common issues, and support your application in Docker on the target. This is not intended as an official document to support all available applications and binaries in the file system. Solutions provided for the use cases in this topic might not be applicable to other applications and binaries and might warrant further investigation and debugging by the developer.
- > Vendors might sometimes provide support for Docker with their applications and libraries. Find out from your vendors if such support exists for your use case on the NVIDIA DRIVE® OS Linux Guest VM.

The `sample_hello_world` application will be used to showcase ways to investigate some of these common issues, as well as identify and resolve requirements to support running your applications on Docker within the NVIDIA DRIVE OS Linux Guest VM.

This topic assumes that the DriveWorks sample is available on the target file system and has been compiled. If not, see the *DriveWorks SDK Reference Documentation* on how to add DriveWorks and the sample to the target file system.

Identifying Dependencies and Requirements

To identify dependencies and requirements, use the `ldd` tool.

With `ldd`, you will get a list of the shared object dependencies that your compiled application will require. Run `ldd` against the `sample_hello_world` application and view the output.

```
$ ldd sample_hello_world
```

Edit the `drivers.csv` file to include the shared library paths from the `ldd` output that are not present in `drivers.csv`. Be aware that some paths might be symlinks. Identify what the symlinks resolve to and also include those paths in the `drivers.csv` file.



Note: For symlinks, include the entry as `sym, <path>` rather than `lib, <path>`.

After adding the required libraries and symlinks, the `drivers.csv` file should look similar to the following:

```
dir, /usr/lib/firmware/tegra23x

lib, /usr/lib/libcuda.so.1
lib, /usr/lib/libnvrn_gpu.so
lib, /usr/lib/libnvrn_mem.so
lib, /usr/lib/libnvrn_sync.so
lib, /usr/lib/libnvrn_host1x.so
lib, /usr/lib/libnvos.so
lib, /usr/lib/libnvsocsys.so
lib, /usr/lib/libnvtegrahv.so
lib, /usr/lib/libnvscipc.so
lib, /usr/lib/libnvrn_chip.so
lib, /usr/lib/libnvcucompat.so

lib, /lib/aarch64-linux-gnu/libEGL_nvidia.so.0
sym, /usr/lib/libcuda.so
lib, /usr/lib/libcuda.so.1
sym, /usr/lib/libnvscibuf.so
lib, /usr/lib/libnvscibuf.so.1
lib, /usr/lib/libnvscicommmon.so.1
lib, /usr/lib/libnvscipc.so
lib, /lib/aarch64-linux-gnu/libudev.so.1
lib, /lib/aarch64-linux-gnu/libusb-1.0.so.0
lib, /lib/aarch64-linux-gnu/librt.so.1
lib, /lib/aarch64-linux-gnu/libX11.so.6
lib, /lib/aarch64-linux-gnu/libXrandr.so.2
lib, /lib/aarch64-linux-gnu/libXinerama.so.1
lib, /lib/aarch64-linux-gnu/libXi.so.6
lib, /lib/aarch64-linux-gnu/libXcursor.so.1
lib, /usr/lib/libdrm.so.2
lib, /lib/aarch64-linux-gnu/libdl.so.2
lib, /lib/aarch64-linux-gnu/libpthread.so.0
lib, /lib/aarch64-linux-gnu/libXext.so.6
lib, /lib/aarch64-linux-gnu/libXxf86vm.so.1
lib, /lib/aarch64-linux-gnu/libGLESv2_nvidia.so.2
lib, /lib/aarch64-linux-gnu/libstdc++.so.6
lib, /lib/aarch64-linux-gnu/libm.so.6
lib, /lib/aarch64-linux-gnu/libgcc_s.so.1
lib, /lib/aarch64-linux-gnu/libc.so.6
lib, /usr/lib/libgnat-23.20220512.so
lib, /usr/lib/libnvrn_host1x.so
lib, /usr/lib/libnvda_runtime.so
lib, /usr/lib/libnvidia-glsi.so.535.00
lib, /usr/lib/libnvrn_chip.so
```

```
lib, /usr/lib/libnvrm_surface.so
lib, /usr/lib/libnvrm_sync.so
lib, /usr/lib/libnvos.so
lib, /usr/lib/libnvrm_gpu.so
lib, /usr/lib/libnvrm_mem.so
lib, /usr/lib/libNvFsiCom.so
lib, /usr/lib/libnvmedia_iep_sci.so
lib, /usr/lib/libnvmedia2d.so
lib, /usr/lib/libnvmedialdc.so
lib, /usr/lib/libnvmedia_ijpe_sci.so
lib, /usr/lib/libnvmedia_ide_parser.so
lib, /usr/lib/libnvmedia_ide_sci.so
lib, /usr/lib/aarch64-linux-gnu/libz.so.1
lib, /usr/lib/libnvmedia_tensor.so
lib, /usr/lib/libnvmedia_dla.so
lib, /usr/lib/libnvscistream.so.1
lib, /usr/lib/aarch64-linux-gnu/libgomp.so.1
lib, /usr/lib/libnvsipl.so
lib, /usr/lib/libnvsipl_devblk.so
lib, /usr/lib/libnvsipl_query.so
lib, /usr/lib/libnvparsers.so
lib, /usr/lib/aarch64-linux-gnu/libnvinfer.so.8
lib, /usr/lib/aarch64-linux-gnu/libnvonnxparser.so.8
lib, /usr/lib/aarch64-linux-gnu/libnvinfer_plugin.so.8
lib, /usr/lib/aarch64-linux-gnu/libcudnn.so.8
lib, /usr/lib/aarch64-linux-gnu/libcudnn_adv_infer.so.8
lib, /usr/lib/aarch64-linux-gnu/libcudnn_adv_train.so.8
lib, /usr/lib/aarch64-linux-gnu/libcudnn_cnn_infer.so.8
lib, /usr/lib/aarch64-linux-gnu/libcudnn_cnn_train.so.8
lib, /usr/lib/aarch64-linux-gnu/libcudnn_ops_infer.so.8
lib, /usr/lib/aarch64-linux-gnu/libcudnn_ops_train.so.8
lib, /lib/aarch64-linux-gnu/libxcb.so.1
lib, /lib/aarch64-linux-gnu/libXrender.so.1
lib, /lib/aarch64-linux-gnu/libXfixes.so.3
lib, /usr/lib/libnvpaintf.so
sym, /lib/aarch64-linux-gnu/libGL.so
sym, /lib/aarch64-linux-gnu/libGL.so.1
lib, /lib/aarch64-linux-gnu/libGL.so.1.7.0
lib, /lib/aarch64-linux-gnu/libGLX.so.0
lib, /lib/aarch64-linux-gnu/libGLdispatch.so.0
lib, /lib/aarch64-linux-gnu/libGLU.so.1
lib, /usr/lib/libnvscsys.so
lib, /usr/lib/libnvidia-rmapi-tegra.so.535.00
lib, /usr/lib/libnvrm_interop_gpu.so
lib, /usr/lib/libnvtegrahv.so
lib, /usr/lib/libnvivc.so
lib, /usr/lib/libnvscievent.so
lib, /usr/lib/libnvvideo.so
lib, /usr/lib/libnvvic.so
lib, /usr/lib/libnvmedia_eglstream.so
lib, /usr/lib/libnvfusacap.so
lib, /usr/lib/libnvsipl_control.so
lib, /usr/lib/libnvsipl_devblk_cdi.so
lib, /usr/lib/libnvsipl_devblk_ddi.so
```

```

lib, /usr/lib/libnvsipl_devblk_crypto.so
lib, /usr/lib/libnvda_compiler.so
lib, /lib/aarch64-linux-gnu/libXau.so.6
lib, /lib/aarch64-linux-gnu/libXdmcp.so.6
lib, /usr/lib/libnvpaumd.so
lib, /usr/lib/libnvrm_stream.so
lib, /usr/lib/libnvisppg.so
lib, /usr/lib/libnvpkcs11.so
lib, /lib/aarch64-linux-gnu/libbsd.so.0
lib, /usr/lib/libnvisp.so
lib, /usr/lib/libteec.so
lib, /usr/lib/libnvvse.so
sym, /usr/lib/libnvscisync.so
lib, /usr/lib/libnvscisync.so.1
lib, /usr/lib/libnvcula.so
lib, /usr/lib/libnvidia-eglcore.so.535.00
lib, /usr/lib/libnvdc.so
lib, /usr/lib/libnvimp.so
lib, /usr/lib/libnvddk_2d_v2.so
lib, /usr/lib/libnvddk_vic.so

```

If your resulting `drivers.csv` file content does not look like the preceding sample output, copy this and replace the entirety of the `drivers.csv` file with this content.

You are now ready to run the `sample_hello_world` application in Docker. If you have not already, change directory to the location of the sample application and run the following command.

```
$ sudo docker run --rm --runtime nvidia --gpus all -v /usr/local/driveworks-5.12:/usr/local/driveworks -v /usr/local/cuda-11.4:/usr/local/cuda -v $(pwd):$(pwd) -w $(pwd) --security-opt systempaths=unconfined ubuntu:20.04 ./sample_hello_world
```

Some specific explanations about the preceding Docker command might be warranted.

- Using `ldd` will list a number of libraries from the `/usr/local/cuda` and `/usr/local/driveworks` paths. However, these paths are symlinks, and they will not resolve to the actual directory paths `/usr/local/cuda-11.4` and `/usr/local/driveworks-5.12`. The CSV will mount them appropriately, but the sample binary will complain that it cannot find them because the symlink that resolves them is not in the image. To address this, you must explicitly mount the CUDA and DW paths from the host to what they would have been with symlinks.

```

-v /usr/local/cuda-11.4:/usr/local/cuda
-v /usr/local/driveworks-5.12:/usr/local/driveworks

```

This addresses a number of the dependencies within those two directories and allows you to remove them from the CSV.

- The sample requires some access to the host `/proc` paths. This is identified by running `strace` against the application to examine the files it tries to open. However, these paths cannot be mounted with the CSV as Docker dynamically allocates a namespace separate from the host.

Normally, the paths can be included in Docker by specifying the `--privileged` argument, but that is too permissive. Rather, it is recommended to use the `--`

`securit-opt systempaths=unconfined` argument. It enables access to `/proc`, but it is not as permissive as `--privileged`. However, depending on the use case of your application, you might require more permissions to access resources on the host. In that case, `--privileged` will do fine.

You have successfully identified dependencies and supported another application to run via Docker on the NVIDIA DRIVE OS Linux Guest VM. Continue to follow the template already present in those files while making necessary changes to support your other applications.

Chapter 7. System Software Components and Interfaces

This topic provides guidance on using system software and component interfaces to interact with the platform. Examples of platform interfaces are file system components, I2C drivers, and SPI drivers.

7.1 Calculating GPIO Index in Linux

The following section describes how to calculate the GPIO index:

1. The Orin GPIOs are grouped in banks and each bank has up to 8 pins. The bank and the pin information of a GPIO is fundamental for calculating its index.

GPIO is represented as X.Y, where:

- X encodes the bank and can have the following values: PA, PB, ..., PY or PAA, ... PFF as listed in table below..

For the banks with naming scheme P*, the "P" must be ignored, i.e., "PA" refers to bank A.

- Y encodes the pin within a bank. For example, 00 for pin 0, 01 for pin 1, ..., and 07 for pin 7.

2. The banks are associated to a base and can also be sequentially translated into a bank_index as shown in the table below:

Bank	AA	BB	CC	DD	EE	GG	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q	R	X	Y	Z	AC	AD	AE	AF	AG
Bank_start_index	0	8	12	20	23	31	0	8	9	17	21	29	35	43	51	58	64	72	76	84	92	100	108	114	122	130	138	146	150	152	156
Base	316 (tegra234-gpio-aon)																														

The base numbering may change in cases where customization is done in kernel and can be identified using the dmesg log:

```
nvidia@tegra-ubuntu:~$ cat /sys/kernel/debug/gpio | grep gpiochip
[ 9.965908] gpiochip0: registered GPIOs 348 to 511 on tegra234-gpio
[ 9.970024] gpiochip1: registered GPIOs 316 to 347 on tegra234-gpio-aon
```

The base is 348 for tegra234-gpio and 316 for tegra234-gpio-aon from the above log.

3. The GPIO index can be calculated as: GPIO_index = base + bank_start_index + pin.

The following is an example of calculating the GPIO index for PS.05.

- a. PS.05 represents bank S and pin 5.
- b. Bank S has bank_index 113 and base 348(tegra-gpio base)
- c. GPIO index is 466 (466 = 348 + 113 + 5)

7.2 Persistence Across Bootburn Flashing Using Persistent Partition

NVIDIA DRIVE® OS provides persistence across bootburn flashing for user data and metadata partitions. The persistence works by reserving a partition in bootburn that is not written or modified in the flashing process. Due to this effect, the data in this partition is completely unchanged across bootburn flashing. After the filesystem is loaded, the user data partition is mounted on /home, and the persistent metadata is overlayed on /etc.

7.2.1 Persistent Data Across Flashing

The following data is persistent across flashing:

1. SSH host keys uniquely identify the system for all SSH connections.
2. User account metadata files containing details and properties of each user account and group.
 - > /etc/passwd: List of the system and local users.
 - > /etc/group: List local and system groups and user account membership information.
 - > /etc/gshadow and /etc/shadow: Contain user account and hashed-password (using crypt).
 - > /etc/subgid and /etc/subuid: The range of group-ids and user-ids allowed for local user account and groups.

7.2.2 Why You Require Persistent Data Across Flashing

Flashing the rootfs partition to update the filesystem must not change or erase the system identity, user account information, or user data. The persistent partitions store user data, metadata, and SSH-host keys (the system's identity for SSH connections). Because the persistent partitions (as the name suggests) are not changed by bootburn flashing or drive update, persistent partitions are required to store such data across flashing.

7.2.3 Workflow with Persistent Partition

The persistent partition workflow consists of the flashing phase and the filesystem phase.

Flashing

1. If the bootburn utility is executed, the filesystem image is always flashed to the filesystem partition.
2. If the bootburn utility is executed with the `--init-persistent-partitions` option, the prebuilt user metadata and data images are flashed to persistent partitions (as specified in PCT configuration in `global_storage.cfg`).
3. If the bootburn utility is executed without the `--init-persistent-partitions` option, there are no changes to persistent partitions.

Filesystem Boot

1. Filesystem mounts the existing user metadata and data persistent partitions.
2. Filesystem checks for the stamp `/etc/nvidia/skip-oem-config.stamp`.
 - > If yes, proceeds to the next step.
 - > If no, OEM-config is executed to set up user accounts.

After oem-config is executed, the stamp `/etc/nvidia/skip-oem-config.stamp` is added, and this is written to persistent metadata partition.
3. Check if `/opt/drive-linux-first-boot.stamp` exists.
 - > If yes, applies fix-up for persistent partition.
 - > If no, proceeds to the next step.

After the fix-up is complete, `/opt/drive-linux-first-boot.stamp` is deleted.
4. Filesystem proceeds to reach the command-line login prompt.

Default Behaviors in NVIDIA DRIVE OS Linux

1. When the bootburn utility is run with the `--init-persistent-partitions` option:
 - a. The `/etc/nvidia/skip-oem-config.stamp` should not exist.
 - i. The OEM-config is executed to set up user accounts.
 - ii. After oem-config is executed, the stamp `/etc/nvidia/skip-oem-config.stamp` is added, and this is written to persistent metadata partition.
 - b. The fix-up is executed (as the filesystem image contains the stamp `/opt/drive-linux-first-boot.stamp`) but is equivalent to a no-op.
2. When the bootburn utility is run without the `--init-persistent-partitions` option:
 - a. `/etc/nvidia/skip-oem-config.stamp` should already exist and oem-config is skipped.
 - b. The fix-up is executed, and this cleans up persistent metadata partition by removing everything other than the necessary files, such as `/etc/passwd|group|shadow|gshadow|subuid|subgid`, `/etc/ssh/ssh_host_key*`, and `/etc/nvidia/<oem-config-stamps>`.

- > Cleanup is required because all files added /etc go to the persistent partition (even if they apply to the rootfs partition) due to overlayfs limitation.



Note: These are the default behaviors in NVIDIA DRIVE OS Linux. However, filesystem users can always change the behavior by removing or adding stamps again (/etc/nvidia/skip-oem-config.stamp or /opt/drive-linux-first-boot.stamp).

7.2.4 Methods to Reset Persistent Partition

Use bootburn with additional argument --init_persistent_partitions to reset the persistent partition.

Alternatively, if you are using `create_bsp_images.py` and `flash_bsp_images.py`, add `--init_persistent_partitions` to only `create_bsp_images.py` and `flash_bsp_images.py` resets the persistent partition.

To reset the partition there are two (2) different workflows:

1. Using `bootburn.py` to flash target: Please add `--init_persistent_partitions` command-line argument to `bootburn`.
2. Using `create_bsp_images.py` + `flash_bsp_images.py` to flash target: Please add `--init_persistent_partitions` to `create_bsp_images.py`. (No `--init_persistent_partitions` command-line argument required for `flash_bsp_images`.)

7.2.5 Data Migration for Persistent Partitions

This release introduces an enhanced persistent partition workflow where persistent partition images are created by `build-fs` and flashed by the `bootburn` utility if `--init-persistent-partitions` is used. The work in the current release is not compatible with the previous releases of NVIDIA DRIVE OS Linux including 6.0.4.0. If you have 6.0.4.0 filesystem flashed, updating the filesystem only updates the rootfs, but the data (although not erased) in the persistent partition is not usable for the current release. You can use the following steps to migrate your persistent data across releases.

Moving Data from Version 6.0.4.0 or Earlier to the Current Version

> Setting Up to Migrate Data

1. Enable a WAR in `/usr/sbin/nv_init.sh` script before it launches `driveos-persistence.sh`.

```
#!/bin/bash
...
export PS4='+(${BASH_SOURCE} :${LINENO}): ${FUNCNAME[0]}:+${FUNCNAME[0]}()'
/bin/bash # <== Please add root shell HERE
# Persistence service setup
/bin/bash $DBG_OPTS /usr/sbin/driveos-persistence.sh
sync
....
```

```
systemctl --no-block isolate graphical.target || true
```

2. Reboot the system.
3. Wait for the root shell by pressing ENTER until you see the root prompt.

> **Migrating Persistent Metadata Partition**

1. Mount the persistent metadata partition.

```
mkdir -p /tmp/tmp_mdata
mount /dev/vblkdev1 /tmp/tmp_mdata
mv /tmp/tmp_mdata/driveos/security/etc/ /tmp/tmp_mdata/
```

2. Remove the old persistent partition directories after migration.

```
rm -rf /tmp/tmp_mdata/driveos/security/
```

3. Unmount the partition.

```
umount /tmp/tmp_mdata
```

> **Migrating Persistent Data Partition**

1. Format data partition.

```
mkfs.ext4 /dev/vblkdev3 # (input y if any prompt comes up)
```

2. Mount the persistent data partition.

```
mkdir -p /tmp/tmp_data
mount /dev/vblkdev3 /tmp/tmp_data
```

3. Move the data in the home directory.

```
mv /home /tmp/tmp_data/
```

4. Unmount the partition.

```
umount /tmp/tmp_data
```

> **After Migration**

- Flash the NVIDIA DRIVE® platform with the 6.0.5.0 SDK.

Moving Data from Current Version to Version 6.0.4.0 or Earlier

> **Setting Up to Migrate Data**

1. Enable a WAR in /usr/sbin/nv_init.sh script before it launches driveos-persistence.sh.

```
#!/bin/bash
...
export PS4='+(${BASH_SOURCE} :${LINENO}): ${FUNCNAME[0]}:+${FUNCNAME[0]}()(): '
/bin/bash # <== Please add root shell HERE

# Persistence service setup
/bin/bash $DBG_OPTS /usr/sbin/driveos-persistence.sh
sync
....
systemctl --no-block isolate graphical.target || true
```

2. Reboot the system.

3. Wait for the root shell by pressing ENTER until you see the root prompt.

> **Migrating Persistent Metadata Partition**

1. Mount the persistent metadata partition.

```
mkdir -p /tmp/tmp_mdata
```

```
mount /dev/vblkdev1 /tmp/tmp_mdata
mkdir -p /tmp/tmp_mdata/driveos/security/
mv /tmp/tmp_mdata/etc/ /tmp/tmp_mdata/driveos/security/
```

2. Remove the old persistent partition directories after migration.

3. Unmount the partition.

```
umount /tmp/tmp_mdata
```

> Migrating Persistent Data Partition

To prepare the system persistent data migration,

1. Flash NVIDIA DRIVE platform with 6.0.4.0 or an earlier version.
2. Add a WAR in nv_init.sh to execute /bin/bash before running driveos-peristence.sh.
3. Reboot the system.
4. Wait for the root shell by pressing ENTER until you see the root prompt.

To migrate the data,

1. Mount the persistent data partition.

```
mkdir -p /tmp/tmp_data
mount /dev/vblkdev3 /tmp/tmp_data
```

2. Move the data in the home directory to the rootfs partition.

```
mv /tmp/tmp_data/home/ /
```

3. Unmount the partition.

```
umount /tmp/tmp_data
```

4. Remove WAR by deleting the line and adding the /bin/bash for the root shell.

> After Migration

- Reboot the system.

7.3 Version Checker

NVIDIA® DRIVE OS Linux contains many software components with packages installed. The NVIDIA Orin Devkit hardware includes many modules having firmware flashed. We provide the version checker tool to ensure correctness/checks of software components and the flashed firmwares. This tool provides a holistic way to check a list of firmware or filesystem packages using a data-driven approach. The version checker is divided into the frontend root node and the backend leaf nodes. The root node version checker manages passing board info, logging, filters output messages to the shell (standard output & /dev/console), iteratively executes the leaf node version checkers, and manages lower-layers errors. The version checker tool is modeled based on a tree with root node version checker responsible for all leaf node version checkers. The root node version checker uses root node config, and each leaf node version checker uses the respective leaf node config, both of which use the YAML format.

The root node version checker does the following to execute different leaf node modules:

- > Root node version checker reads root node config, obtains board-id, and makes the list of leaf node instances to call.

- > For each leaf node firmware, root node version checker passes board-id, calls leaf version checker with, and passes leaf node config.

The steps used to check the version using the leaf node version checker are the following:

- > If Leaf node config uses *type=debian* then:
 - Leaf node version checker gets the expected manifest of package versions from the config.
 - Leaf node version checker uses dpkg-query (reference Tool for debian packages) to get the packages and their versions in the filesystem.
 - Leaf node version checker then tallies manifest packages to filesystem packages and where packages tally, compares versions of packages.
 - Leaf node version checker depending on MATCH/MISMATCH, prints the PASS/FAIL result.
- > If Leaf node config uses *type=firmware*, then:
 - Leaf node version checker gets the expected firmware version based on the config file.
 - Leaf node version checker uses the Tool attribute and executes the command to fetch firmware version from the board.
 - Leaf node version checker compares expected firmware version to obtained firmware version and prints the PASS/FAIL result.

Version Checker Prerequisites

Version checker requires the following setup commands to be executed one time on the target before running it. Ensure that you have completed the target-side setup and have logged in using your username and password in the minicom console.

1. Version checker in the default configuration prints all output to standard output and prints just error messages to /dev/console.
2. Run the version checker application as shown in the command line below.

Notes specific to different filesystems:

- > Filesystems driveos-oobe-rfs and driveos-oobe-desktop-rfs:
 1. Using the preinstalled ssh server, connect from host to target using the ssh shell.
 2. Run the version checker in the ssh shell.
 3. The version checker prints to standard output shall be seen in the current shell, and the prints to /dev/console should be seen in the minicom console.
- > Filesystem driveos-core-rfs:
 1. In this filesystem, the ssh server package is not preinstalled, so it is not possible to start an ssh shell from the host to the target is possible.
 2. Use the minicom console to run the version checker.
 3. The version checker will print messages from standard output and /dev/console to the minicom console.

Version Checker Command Line

Version checker is always executed using the root version checker and uses the following command-line as shown below.

```
usage: version_checker_root.py [-h] [-i YAML_PATH] [-l LOGFILE_PATH] [-v]
Checks all system firmwares.
-h, --help            show this help message and exit
-i YAML_PATH, --input-yaml YAML_PATH
                      Input YAML config file.
-l LOGFILE_PATH, --log-file-path LOGFILE_PATH
                      Optional Argument: Log file path. Default path is /var/log/
version_checker.log.
-v, --verbose          Optional Argument: Verbose logging i.e., print info/debug
messages to /dev/console.
```

Examples running `version_checker_root.py` application:

1. Run with default log file path (`/var/log/version_checker.log`) and input config `/etc/nvidia/version_checker/manifest/version_data_root.yml`
 - a. Prints info messages to standard output, log file, and only errors to `/dev/console`.
`version_checker_root.py -i /etc/nvidia/version_checker/manifest/
version_data_root.yml`
2. Run with custom log file path `<log_file>` and same input config from step 1.
 - a. Prints info messages to standard output, log file, and only errors to `/dev/console`.
`version_checker_root.py -i /etc/nvidia/version_checker/manifest/
version_data_root.yml -l <log_file>`
3. Run with default log file path (`/var/log/version_checker.log`) and same input config from step 1 in verbose mode.
 - a. Prints info messages to standard output, to log file, and to `/dev/console`.
`version_checker_root.py -i /etc/nvidia/version_checker/manifest/
version_data_root.yml -v`

Config Files

The root node config contains the following fields per child node:

- Path to child node version checker.
- Config file of child node version checker.
- The version of DRIVE OS LINUX SDK.
- Path of DRIVE OS LINUX SDK manifest.

Leaf node config contains the following fields:

- Name of the module like filesystem or firmware.
- Type of the module: *debian* or *firmware*.
- For *type=firmware*, we have:
 - The firmware version flashed on board.
 - Firmware version list corresponding to board.
 - List of compatible boards (i.e., boards containing the firmware and has its version retrievable) corresponding to the firmware.

- > For *type=debian* we have:
- Tool to fetch package versions in the filesystem.
 - Path of filesystem manifest.

The below examples provide a root node config and a leaf node config.

Root node config:

```
sdk_version: "<release>-<GCID>"
sdk_manifest: "/etc/nvidia/version-ubuntu-rootfs.txt"
versions:
  - versions: ["/usr/sbin/version_checker_leaf.py", "/etc/nvidia/version_checker/
manifest/rootfs_debians.yml"]
  - versions: ["/usr/sbin/version_checker_leaf.py", "/etc/nvidia/version_checker/
manifest/ethernet_firmware_1.yml"]
```

Leaf node config:

1. Example with *type=debian*:

```
sdk_version: "<release>-<GCID>"
type: "debian"
filesystem_packages:
  name: "Filesystem debian packages"
  tool: "dpkg-query -W -f '${Package}=${Version}\n'"
  manifest: "/etc/nvidia/rootfilesystem-manifest/driveos-rfs.MANIFEST.json"
```

2. Example *type=firmware*:

```
sdk_version: "<release>-<GCID>"
type: "firmware"
firmware:
  name: "8 Port 100Mb Ethernet Switch"
  tool: "/bin/bash -c '/lib/firmware/marvell_ethernet/common/ota/linux/nvidia/update-
firmware.sh --printversion | grep \"Current firmware version flashed is\" | cut -d\"
\" -f7'"
  firmware_versions: # <list of dicts>
    - compatible_boards: [ "e3550_t194b", "e3550_t194a" ]
      firmware_version: ["02.07.1001"]
```

Output Messages and Logging

The version checker has the following structure for outputting messages into standard output, minicom console, and the (persistent) log file. The minicom console output messages appear only if an error occurs. The standard output and the log file contain informational messages to help users understand or debug tool/config files. The verbose ensures that all of the minicom console, standard output, and log files contain all the informational messages.

Sample /dev/console output on an error:

```
*** Version mismatch on firmware "8 Port 100Mb Ethernet Switch", expected ['02.07.1000']
but found 02.07.1001. ***
*** Error version check for firmware "8 Port 100Mb Ethernet Switch". ***
*** Version mismatch on firmware "8 Port 100Mb Ethernet Switch - Secure Keys", expected
['02.07.1000'] but found 02.07.1001. ***
*** Error version check for firmware "8 Port 100Mb Ethernet Switch - Secure Keys". ***
```

Sample standard output as well as log file content /dev/console output on an error:

```
=====
ROOT VERSION CHECKER
=====
Invoking python3 /lib/firmware/version_checker_leaf.py -i /lib/firmware/ethernet/
ethernet_firmware_1.yml -b e3550_t194a

Start checking version of "8 Port 100Mb Ethernet Switch".
Invoking /bin/bash -c '/lib/firmware/marvell_ethernet/common/ota/linux/nvidia/update-
firmware.sh --printversion | grep "Current firmware version flashed is" | cut -d" " -f7'

*** Version mismatch on firmware "8 Port 100Mb Ethernet Switch", expected ['02.07.1000']
but found 02.07.1001. ***
*** Error version check for firmware "8 Port 100Mb Ethernet Switch". ***
Finished checking version of "8 Port 100Mb Ethernet Switch".

Invoking python3 /lib/firmware/version_checker_leaf.py -i /lib/firmware/ethernet/
ethernet_firmware_2.yml -b e3550_t194a

Start checking version of "8 Port 100Mb Ethernet Switch - Secure Keys".
Invoking /bin/bash -c '/lib/firmware/marvell_ethernet/common/ota/linux/nvidia/update-
firmware.sh --printversion | grep "Current firmware version flashed is" | cut -d" " -f7'

*** Version mismatch on firmware "8 Port 100Mb Ethernet Switch - Secure Keys", expected
['02.07.1000'] but found 02.07.1001. ***
*** Error version check for firmware "8 Port 100Mb Ethernet Switch - Secure Keys". ***
Finished checking version of "8 Port 100Mb Ethernet Switch - Secure Keys".
```

Version Checker API Documentation

The version checker supports the following semantics below for each root node and leaf node version checker.

Root version checker semantics:

Field	Type	Acceptable Values	Required	Syntax	Instructions
sdk_version	str	SDK version in form of A.B.C.D (like 5.2.6.0, 6.0.0.0, etc.)	Yes	"sdk_version" : "<sdk_ver_num>"	Please add SDK release version info and must match the value on file in sdk_manifest.

Field	Type	Acceptable Values	Required	Syntax	Instructions
sdk_manifest	str	Path of the file containing SDK version.	Yes	"sdk_version": <file path having sdk version>	Please add the path of the file containing the SDK version to the value of key <i>sdk_version</i> .
versions	list(dict)	Contains the list dictionaries in the form, {version: <version data>}	Yes	"versions" : <list of dict>; dict is version: <version data>	Please add a list of version entries under the version key.
version	list(str)	2 element list of strings ["leaf checker", "leaf config"]	Yes	"version" : ["leaf checker", "leaf config"]	Please add <path_to_firmware_checker, path_to_firmware_data> in the list for each.

Leaf version checker semantics:

Field	Type	Acceptable Values	Required	Syntax	Instructions
sdk_version	str	SDK version in form of A.B.C.D (like 5.2.6.0, 6.0.0.0, etc.)	Yes	"sdk_version" : "<sdk_ver_num>"	Please add SDK release version info and must match the value on file in <i>sdk_manifest</i> .
type	str	Type of config, either "debian" or "firmware".	Yes	"type": <type>	Please add <type> of config which is "debian" or "firmware".
name	str	Name of the version entry.	Yes	"name" : "<version entry name>"	Please add the full name of the version entry in this field. This can be any user-defined name.

Field	Type	Acceptable Values	Required	Syntax	Instructions
tool	str	Command-line tool to query existing version from the system.	Yes	"tool": "<cmdline to query version>"	Please add the full cmdline, including the path to the low-level utility to fetch the version of the component with arguments.
firmware_version	list(dict)	Contains the list of dict in the form, {"compatible_board_ids": <value>, "firmware_version": <value>}	Yes	"firmware_version": <list of dicts>	Please add the firmware definition listing under this field.
compatible_board_ids	list(str)	Contains a list of board-ids.	Yes	"compatible_board_ids": <"board1", "board2", ..>	Please add all supported board_ids containing this firmware.
firmware_versions	list(str)	Contains a list of firmware versions.	Yes	"firmware_versions": <"ver1", "ver2", ..>	Please add all possible firmware versions corresponding to a compatible board.
filesystem_packages	dict(dict)	Contains a dict describing filesystem attributes	Yes	"filesystem_packages": <fs_dict>	Please add dict of filesystem attributes to this field.
manifest	str	A string containing the path of the RFS debian manifest.	Yes	"manifest": <path>	Please add the path of the RFS debian manifest to this field.

Version Checker Developer API Documentation

Version checker tools are implemented in python3 and contains python3 compatible documentation within code. To open the documentation, please use the steps below.

1. `cd <top>/drive-linux/filesystem/contents/bin`
2. `pydoc3 -w ./`
3. Open the generated `version_checker_leaf.html` and `version_checker_root.html` on your browser.

7.4 Linux File Systems

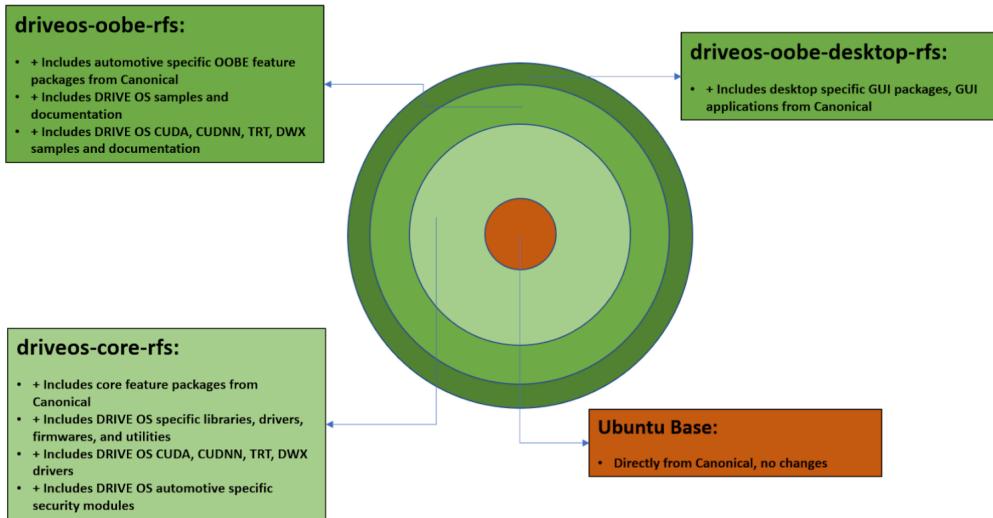
The DRIVE OS Linux consists of a hierarchy of file systems. Each layer of the hierarchy of file systems is assigned a purpose and the particular purpose at that layer defines the contents of the file system.

The following image represents the hierarchy of file systems. Each layer builds on top of other layers.

The base layer at the bottom of the hierarchy is the Canonical (open source) Ubuntu Base file system. The NVIDIA® DRIVE OS Linux file systems are built on top of the Ubuntu Base file system.

The next layer is named driveos-core-rfs and is purposed to contain automotive-specific Debian packages, drivers, libraries, tools, firmwares, scripts, and other necessary files for core functionality. The driveos-core-rfs file system is useful for production systems.

The next layer is named driveos-oobe-rfs and is an Out Of Box Console for developers. It contains automotive-specific samples, documentation, and developer friendly tools. This file system is intended to be used by developers for developers.



The final layer is named driveos-oobe-desktop-rfs and is an Out Of Box GUI Desktop Experience for developers. It contains automotive GUI desktop interface, specific samples, documentation, and developer friendly tools. This file system is intended to be used by developers for software development.

The itemized listing of Debian packages in the DRIVE OS Linux file system is available in an end-user friendly descriptive version alongside the EXT4 image file of the file system (depending on driveos-oobe-rfs, driveos-core-rfs, or driveos-oobe-desktop-rfs) in the following paths:

- > /opt/nvidia/driveos/<release>/<DRIVEOS_GCID>/drive-linux/filesystems/driveos-core-ubuntu-20.04-rfs/driveos-core-ubuntu-20.04-rfs.manifest
- > /opt/nvidia/driveos/<release>/<DRIVEOS_GCID>/drive-linux/filesystems/driveos-oobe-ubuntu-20.04-rfs/driveos-oobe-ubuntu-20.04-rfs.manifest
- > /opt/nvidia/driveos/<release>/<DRIVEOS_GCID>/drive-linux/filesystems/driveos-oobe-desktop-ubuntu-20.04-rfs/driveos-oobe-desktop-ubuntu-20.04-rfs.manifest

Where <DRIVEOS_GCID> is the GCID of DRIVE OS Linux and <release> is the version of NVIDIA DRIVE OS Linux.

Additionally, the file systems also contain a listing of all installed packages (useful for DRIVE OS Build-FS) at (depending on driveos-oobe-rfs, driveos-core-rfs, or driveos-oobe-desktop-rfs):

- > /etc/nvidia/rootfilesystem-manifest/driveos-core-ubuntu-20.04-rfs.MANIFEST.json
- > /etc/nvidia/rootfilesystem-manifest/driveos-oobe-ubuntu-20.04-rfs.MANIFEST.json
- > /etc/nvidia/rootfilesystem-manifest/driveos-oobe-desktop-ubuntu-20.04-rfs.MANIFEST.json

Finally, the file system contains files overlayed over the Debian packages from different modules like drivers, libraries, tools, firmware, scripts, and samples. The set of modules is included in driveos-core-rfs and driveos-oobe-rfs. The itemized listing is available in the [CopyTarget](#) topic.

7.4.1 DRIVE OS Linux File Systems

The following table shows the Debian package to be installed for the specified file system.

File System	File to Download	Purpose
driveos-oobe-ubuntu-20.04-rfs	nv-driveos-linux-driveos-oobe-ubuntu-20.04-rfs-<release>-<DRIVEOS_GCID>_<release>-<DRIVEOS_GCID>.amd64.deb	Intended to be used by developers as a development or reference filesystem. Contains NVIDIA automotive sample applications and corresponding documentations.
driveos-oobe-desktop-ubuntu-20.04-rfs	nv-driveos-linux-driveos-oobe-desktop-ubuntu-20.04-rfs-<release>-<DRIVEOS_GCID>_<release>-<DRIVEOS_GCID>.amd64.deb	Intended to be used by developers as a development or reference filesystem. Contains NVIDIA automotive GUI desktop interface, sample applications, and corresponding documentations.
driveos-core-ubuntu-20.04-rfs	nv-driveos-linux-driveos-core-ubuntu-20.04-rfs-<release>-<DRIVEOS_GCID>_<release>-<DRIVEOS_GCID>.amd64.deb	Intended to be used by production systems. Contains NVIDIA automotive drivers, libraries, tools, firmware, scripts, and other required files for core functionality.

Where <DRIVEOS_GCID> is the GCID of NVIDIA DRIVE® OS Linux and <release> is the NVIDIA DRIVE® OS Linux SDK version.


Note:

- > The DRIVE OS Linux file system Debian is installable only in the local Debian installer workflow. The steps for the local Debian installer are in the Install DRIVE OS Linux Debian Packages topic.
- > To install or get different file systems using SDK Manager or Docker path, see the DRIVE OS Linux SDK Manager or Docker documentation.

7.4.2 Filesystem Manifest

Each filesystem is accompanied by a Filesystem Manifest. The manifest is located with the filesystem at :

```
/opt/nvidia/driveos/<release>/<GCID>/drive-linux/filesystems/<fs_variant>/<fs_variant>.MANIFEST.json
```

Where:

- > <DRIVEOS_GCID> is the GCID of NVIDIA DRIVE OS Linux
- > <release> is the NVIDIA DRIVE OS Linux SDK version
- > <fs_variant> is one of the values from the list: [driveos-core-ubuntu-20.04-rfs, driveos-oobe-ubuntu-20.04-rfs, driveos-oobe-desktop-ubuntu-20.04-rfs]

The manifest can be used to identify the information defined in the NVIDIA DRIVE OS Build-FS config for generating the filesystem. For more information, see [NVIDIA Build-FS](#) documentation.

7.4.2.1 Prompts During Installation of Filesystem

During installation of the filesystem, certain criteria may trigger one or more interactive prompts for you to confirm or specify additional information. When prompts are issued, you must answer to continue.

There are silent install variables that allow non-interactive installation. These silent install variables are specified in the respective tables of each installation method and an explanation to use them is provided. Silent install variables are set as environment variables and if exposed to the installers, are used.

7.4.2.2 Prompts During Installation of Filesystem from Debian

Interactive Prompt	Acceptable Input	Default	Trigger	Instructions	Silent Install Variable Bypass Criteria
Please input NV_WORKSPACE path. NV_WORKSPACE Path where DRIVEOS SDK is installed	Valid Unix directory path, ensure input is terminated with a trailing slash.	-	This prompt will be presented to the user to confirm if the following condition is met unless the silent install variable criteria has been set with the appropriate value. \${NV_WO is not defined in the environment	This prompt is required to be answered to indicate to Debian installation script the location to create filesystem image.	NV_WORKSPACE=<DIRECTORY_PATH> Where <DIRECTORY_PATH> is the valid DRIVE OS LINUX installed directory; ensure this directory path ends with a slash "/".
"<FILEPATH>" already exists, do you wish to replace the img [y/N] ? Where <FILEPATH> is the absolute filepath where the installer is attempting to install filesystem image to	"y" - yes "n" - no	n (no)	This prompt will be presented to the user to confirm if the following condition is met unless the silent install variable criteria has been set with the appropriate value.	This prompt is presented to prevent automated destruction of potential existing data.	NV_OVERWRITE_IMAGE=yes Note: It is irrelevant whether \${NV_WORKSPACE} has been set or not because \${NV_WORKSPACE} will be set by prior prompt (above) if not already.

7.4.3 Unused Upstream Components in DRIVE OS Linux

The DRIVE OS Linux SDK file system (based on Ubuntu-20.04) has components from upstream that are unused and inapplicable for DRIVE OS Linux. The following list shows the components that are unused due to the reasons mentioned below.

- > **Upower:** Upower daemon is used for power management in Linux. The DRIVE OS Linux guest OS does not have control for power management and is controlled by AURIX.
- > **Pulseaudio:** DRIVE OS Linux does not support pulseaudio and has alternatives for audio.
- > **Alsa:** DRIVE OS Linux does not support pulseaudio and has alternatives for audio.
- > **apt-daily:** Apt-daily is disabled in DRIVE OS Linux SDK file systems to prevent automatic updates of packages in the file system because the baseline file system has been quality tested and delivered.
- > **isc-dhcp-server:** DHCP server is disabled because all documented SDK workflows use the target as a DHCP client.
- > **network-manager:** DRIVE OS Linux supports networking using systemd-networkd instead of network-manager.
- > **ondemand.service:** DRIVE OS Linux does not support CPU governor with ondemand policy.

7.4.4 Logging In



Note:

If you experience instability or timeouts with ssh connections to a DRIVE AGX Orin System in the DRIVE AGX Orin System, set `-o ServerAliveInternal` to keep the ssh session live. For example:

```
$ ssh -o ServerAliveInternal=240 nvidia@<Target IP Address>
```

See the standard Linux ssh manpage documentation for more information.

The file systems (drive-oobe-rfs only) support secured shell (ssh) and serial console logins. Use the secure login feature.

7.4.5 DRIVE OS Linux Username and Password

For information about the default DRIVE OS username and password, and how to change the username and password, see [Changing the Default Target Username and Password](#).

7.4.6 Installing Non-Default File Systems

When the DRIVE OS Linux is installed using the steps in the Install DRIVE OS Linux Debian Packages topic, the default file system installed is driveos-core-ubuntu-20.04-rfs. You can install other file system Debian, such as driveos-oobe-desktop-ubuntu-20.04-rfs or driveos-oobe-ubuntu-20.04-rfs, following these steps.

1. To install the driveos-oobe-rfs or driveos-oobe-desktop Debian package, use the following commands:

```
$ export NV_WORKSPACE="/path/where/SDK/needs/is/installed"
$ sudo -E dpkg -i <deb_file>
```

Enter yes to each follow-on dpkg prompt.

The <deb_file> is the file system Debian from the File System column in the table shown in "[DRIVE OS LINUX File Systems](#)".

2. After installing the file system, flash the target using the guidelines in the DRIVE OS Bootburn document.

7.4.7 Network Configuration in NVIDIA Filesystems

The service/daemon that manages/configures the network in NVIDIA filesystems (driveos-core-rfs, driveos-oobe-rfs, driveos-oobe-desktop-rfs) is `systemd-networkd`, which is different from the standard network manager (`NetworkManager`) used in Ubuntu 20.04 by default.

`systemd-networkd` is the only network manager installed. Hence, utilities like `netplan` aren't installed to support both `systemd-networkd` and `NetworkManager`.

7.4.7.1 Configuring Interfaces

To configure a particular interface, you must create a network file/netdev file in `/etc/systemd/network/`.

For example, to configure interface `eth0` with DHCP, create network file `50-wired.network` and place it in `/etc/systemd/network/`:

```
[Match]
Name=eth0
[Network]
DHCP=ipv4
```

For more information on network file semantics, priority, and location, see the [systemd-networkd man page](#).

7.4.8 Rebuilding the File System from ubuntu-base and Local Mirror

DRIVE OS Linux SDK provides the following additional target-specific components below. These components, along with build-fs and copytarget tools, can be used to rebuild the filesystem starting from the ubuntu-base tarball.

1. Canonical ubuntu-base tarball: nv-driveos-linux-ubuntu-20.04-base-*_amd64.deb
2. Canonical arm64 Debian packages: nv-driveos-linux-ubuntu-20.04-arm64-debians-*_amd64.deb
3. NVIDIA CUDA arm64 Debian packages: cuda-repo-ubuntu2004-11-4-local*arm64.deb
4. NVIDIA cuDNN arm64 Debian packages: cudnn-prune-87-repo-ubuntu2004-8-2-local*arm64.deb
5. NVIDIA TensorRT arm64 Debian packages: nv-tensorrt-repo-ubuntu2004-cuda11.4-trt*arm64.deb
6. NVIDIA Mellanox arm64 Debian packages: nv-driveos-linux-mlnx-docker-arm64-debians-*_amd64.deb
7. NVIDIA Docker arm64 Debian packages: nv-driveos-linux-mlnx-docker-arm64-debians-*_amd64.deb
8. NVIDIA DriveWorks arm64 Debian packages: List of Debian packages:
 - a. driveworks-v*_drive-linux-*.deb
 - b. driveworks_cgf-v*_drive-linux-*.deb
 - c. driveworks_cgf_cross-v*_drive-linux-*.deb
 - d. driveworks_cgf_samples-v*_drive-linux-*.deb
 - e. driveworks_cross-v*_drive-linux-*.deb
 - f. driveworks_data-v*_drive-linux-*.deb
 - g. driveworks_samples-v*_drive-linux-*.deb
 - h. driveworks_stm-v*_drive-linux-*.deb
 - i. driveworks_stm_cross-v*_drive-linux-*.deb
 - j. driveworks_stm_samples-v*_drive-linux-*.deb
9. NVIDIA DRIVE OS Core arm64 Debian packages - List of Debian packages:
 - a. nv-driveos-linux-aurix_*_arm64.deb
 - b. nv-driveos-linux-firmware_*_arm64.deb
 - c. nv-driveos-linux-headers_*_arm64.deb
 - d. nv-driveos-linux-kernel-modules_*_arm64.deb
 - e. nv-driveos-linux-libraries_*_arm64.deb
 - f. nv-driveos-linux-samples_*_arm64.deb
 - g. nv-driveos-linux-security_*_arm64.deb
 - h. nv-driveos-linux-tools_*_arm64.deb
 - i. nv-driveos-linux-core_*_arm64.deb
 - j. nv-driveos-linux-oobe_*_arm64.deb

The following steps are prerequisites and must be performed before executing the steps to rebuild the filesystem below:

1. Ensure that DRIVE OS Linux SDK is installed as per the Getting Started page.
2. Install the build-fs and Copytarget Debian packages to use these tools.
3. Ensure the NV_WORKSPACE shell variable is set and points to the top directory where DRIVE OS Linux SDK is installed.
4. Keep the DRIVE OS Linux SDK Debian packages in \$NV_WORKSPACE and switch to the following directory:

```
cd $NV_WORKSPACE
```

Steps to rebuild a filesystem from ubuntu-base

The rebuilding a filesystem requires the use of a local mirror from the target-specific components above. Execute the steps below to set up the local mirror.

1. Install the driveos-oobe-desktop-rfs SDK package to install its manifest file driveos-\${FS_VARIANT}*MANIFEST.json.

```
$ sudo -E dpkg -i ./nv-driveos-linux-driveos-oobe-desktop-ubuntu-20.04-rfs-*_amd64.deb
```

Here, \${FS_VARIANT} is the name of filesystem variant, e.g. oobe-desktop.

For an example to rebuild driveos-oobe-desktop-rfs, please refer to Example: Steps to rebuild driveos-oobe-desktop-rfs filesystem from ubuntu-base below.

2. Install Canonical ubuntu-base and arm64 Debian SDK packages:

```
$ sudo -E dpkg -i ./nv-driveos-linux-ubuntu-20.04-arm64-debians-<release>-<GCID>_<release>-<GCID>_amd64.deb ./nv-driveos-linux-ubuntu-20.04-base-<release>-<GCID>_<release>-<GCID>_amd64.deb
```

3. Install NVIDIA Mellanox and Docker arm64 Debian packages:

```
$ sudo -E dpkg -i ./nv-driveos-linux-mlnx-docker-arm64-debians--<release>-<GCID>_<release>-<GCID>_amd64.deb
```

4. Copy the NVIDIA CUDA, cuDNN, TensorRT and DriveWorks arm64 Debian packages to \$NV_WORKSPACE/drive-linux/filesystem/contents/debians/nvidia/.

5. Import CUDA bits exported variables by sourcing versions using cmd below:

```
$ source ${NVWORKSPACE}/drive-linux/filesystem/contents/debians/versions.conf
```

6. Build the final filesystem starting from ubuntu-base:

```
$ sudo -E /opt/nvidia/driveos/common/filesystems/build-fs/17/bin/build_fs.py -w ${NV_WORKSPACE}/ -i ${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-${FS_VARIANT}-ubuntu-20.04-rfs.MANIFEST.json -o $PWD/output/
```

Example: Steps to rebuild driveos-oobe-desktop-rfs filesystem from ubuntu-base

The rebuilding use case/example requires the use of a local mirror from the target-specific components above. Execute the steps below to set up the local mirror.

1. Install the driveos-oobe-desktop-rfs SDK package to install its manifest file driveos-oobe-desktop*MANIFEST.json.

```
$ sudo -E dpkg -i ./nv-driveos-linux-driveos-oobe-desktop-ubuntu-20.04-rfs-*_amd64.deb
```

2. Install Canonical ubuntu-base and arm64 Debian SDK packages:

```
$ sudo -E dpkg -i ./nv-driveos-linux-ubuntu-20.04-arm64-debians-<release>-
<GCID>_<release>-<GCID>_amd64.deb ./nv-driveos-linux-ubuntu-20.04-base-<release>-
<GCID>_<release>-<GCID>_amd64.deb
```

3. Install NVIDIA Mellanox and Docker arm64 Debian packages:

```
$ sudo -E dpkg -i ./nv-driveos-linux-mlnx-docker-arm64-debians--<release>-
<GCID>_<release>-<GCID>_amd64.deb
```

4. Copy the NVIDIA CUDA, cuDNN, TensorRT and DriveWorks arm64 Debian packages to \${NV_WORKSPACE}/drive-linux/filesystem/contents/debians/nvidia/.

5. Import CUDA bits exported variables by sourcing versions using cmd below:

```
$ source ${NVWORKSPACE}/drive-linux/filesystem/contents/debians/versions.conf
```

6. Build the final filesystem starting from ubuntu-base:

```
$ sudo -E /opt/nvidia/driveos/common/filesystems/build-fs/17/bin/build_fs.py -w
${NV_WORKSPACE} / -i ${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-
oobe-desktop-ubuntu-20.04-rfs.MANIFEST.json -o $PWD/output/
```



Note: To rebuild a filesystem other than driveos-oobe-desktop, provide the corresponding manifest as input instead of driveos-oobe-desktop-ubuntu-20.04-rfs.MANIFEST.json.

Similarly, to install any filesystem manifest, install the corresponding filesystem SDK package.

Rebuilding a Filesystem from ubuntu-base Using NVIDIA DRIVE OS Core CopyTarget YAML Manifests

This section provides instructions rebuild the Linux RFS using the DRIVE OS Core CopyTarget YAML manifests from \${NV_WORKSPACE}/drive-linux/filesystem/copytarget/manifest/*.yaml to copy the Core files instead of obtaining them from the NVIDIA DRIVE OS Core Debian packages.

This is useful when you have modified the Core files on your host and want to copy them to the filesystem without rebuilding the NVIDIA DRIVE OS Core Debian packages.

Note: If you instead wish to add additional files to an already available filesystem, follow the instructions in [How to Add a Single Debian Package and a Single File to the Linux Filesystem](#).

1. Install the rfs SDK package to install its manifest file driveos-\${FS_VARIANT}*MANIFEST.json.

```
$ sudo -E dpkg -i
./nv-driveos-linux-driveos-${FS_VARIANT}-ubuntu-20.04-rfs-
*_amd64.deb
```

Here, \${FS_VARIANT} is the name of filesystem variant, e.g. oobe-desktop.

For an example to rebuild driveos-oobe-desktop-rfs, please refer to Example: Rebuilding driveos-oobe-desktop-rfs Filesystem from ubuntu-base Using NVIDIA DRIVE OS Core CopyTarget YAML Manifests below.

2. Install Canonical ubuntu-base and arm64 Debian SDK packages:

```
$ sudo -E dpkg -i ./nv-driveos-linux-ubuntu-20.04-arm64-debians-*_amd64.deb
```

```
./nv-driveos-linux-ubuntu-20.04-base-*_amd64.deb
```

3. Install NVIDIA Mellanox and Docker arm64 Debian packages:

```
$ sudo -E dpkg -i  
./nv-driveos-linux-mlnx-docker-arm64-debians-*_amd64.deb
```

4. Copy the NVIDIA CUDA, cuDNN, TensorRT and DriveWorks arm64 Debian packages to \${NV_WORKSPACE}/drive-linux/filesystem/contents/debians/nvidia/.

5. Import CUDA bits exported variables by sourcing versions using cmd below:

```
$ set -a  
$ source  
    ${NV_WORKSPACE}/drive-linux/filesystem/contents/debians/  
versions.conf  
$ set +a
```

6. Update \${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-\${FS_NAME}-ubuntu-20.04-rfs.MANIFEST.json to remove DRIVE OS Core Debian packages.

```
$ sed -i  
    '/nv-driveos-linux-\(aurix\|core\|firmware\|headers\|kernel-modules  
\|libraries\|oobe\|samples\|security\|tools\)\.*/d'  
    ${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-  
${FS_VARIANT}-ubuntu-20.04-rfs.MANIFEST.json
```

7. Update \${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-\${FS_NAME}-ubuntu-20.04-rfs.MANIFEST.json to use the DRIVE OS Core CopyTarget YAML manifests. You may update the "CopyTarget" key in the Build-FS manifest manually to include the YAML manifests from \${NV_WORKSPACE}/drive-linux/filesystem/copytarget/manifest/*.yaml or use the following single command (note that this is one single command that can be pasted into your bash terminal):

```
python3 -B - << END  
import json  
from collections import OrderedDict  
bkConfig = OrderedDict()  
manifest ="${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-  
${FS_VARIANT}-ubuntu-20.04-rfs.MANIFEST.json"  
with open(manifest) as f:  
    data = f.read()  
bkConfig = json.loads(data, object_pairs_hook=OrderedDict)  
bkConfig["CopyTargets"] = [  
    "\${COPYTARGETYAML_DIR}/copytarget-configs.yaml",  
    "\${COPYTARGETYAML_DIR}/copytarget-aurix.yaml",  
    "\${COPYTARGETYAML_DIR}/copytarget-aurix.yaml",  
    "\${COPYTARGETYAML_DIR}/copytarget-firmware.yaml",  
    "\${COPYTARGETYAML_DIR}/copytarget-headers.yaml",  
    "\${COPYTARGETYAML_DIR}/copytarget-kernel-modules.yaml",  
    "\${COPYTARGETYAML_DIR}/copytarget-libraries.yaml",  
    "\${COPYTARGETYAML_DIR}/copytarget-samples.yaml",  
    "\${COPYTARGETYAML_DIR}/copytarget-security.yaml",  
    "\${COPYTARGETYAML_DIR}/copytarget-tools.yaml"  
]  
with open(manifest, 'w') as f:  
    f.write(json.dumps(bkConfig, indent=4, sort_keys=False))  
END
```

8. Build the final filesystem starting from ubuntu-base:

```
$ sudo -E /opt/nvidia/driveos/common/filesystems/build-fs/17/bin/build_fs.py -w ${NV_WORKSPACE}/ -i ${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-${FS_VARIANT}-ubuntu-20.04-rfs.MANIFEST.json -o $PWD/output/
```

Example: Rebuilding driveos-oobe-desktop-rfs Filesystem from ubuntu-base Using NVIDIA DRIVE OS Core CopyTarget YAML Manifests

This section provides an example to rebuild the Linux OOB Desktop RFS using the DRIVE OS Core CopyTarget YAML manifests from \${NV_WORKSPACE}/drive-linux/filesystem/copytarget/manifest/*.yaml to copy the Core files instead of obtaining them from the NVIDIA DRIVE OS Core Debian packages.

This is useful when you have modified the Core files on your host and want to copy them to the filesystem without rebuilding the NVIDIA DRIVE OS Core Debian packages.



Note: If you instead wish to add additional files to an already available filesystem, follow the instructions in [How to Add a Single Debian Package and a Single File to the Linux Filesystem](#).

1. Install the driveos-oobe-desktop-rfs SDK package to install its manifest file driveos-oobe-desktop*MANIFEST.json.

```
$ sudo -E dpkg -i  
./nv-driveos-linux-driveos-oobe-desktop-ubuntu-20.04-rfs-*_amd64.deb
```

2. Install Canonical ubuntu-base and arm64 Debian SDK packages:

```
$ sudo -E dpkg -i ./nv-driveos-linux-ubuntu-20.04-arm64-debians-*_amd64.deb  
./nv-driveos-linux-ubuntu-20.04-base-*_amd64.deb
```

3. Install NVIDIA Mellanox and Docker arm64 Debian packages:

```
$ sudo -E dpkg -i  
./nv-driveos-linux-mlnx-docker-arm64-debians-*_amd64.deb
```

4. Copy the NVIDIA CUDA, cuDNN, TensorRT and DriveWorks arm64 Debian packages to \${NV_WORKSPACE}/drive-linux/filesystem/contents/debians/nvidia/.
5. Import CUDA bits exported variables by sourcing versions using cmd below:

```
$ set -a  
$ source  
${NV_WORKSPACE}/drive-linux/filesystem/contents/debians/versions.conf  
$ set +a
```

6. Update \${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-oobe-desktop-ubuntu-20.04-rfs.MANIFEST.json to remove DRIVE OS Core Debian packages.

```
$ sed -i  
' /nv-driveos-linux-\(aurix\|core\|firmware\|headers\|kernel-modules\|  
libraries\|oobe\|samples\|security\|tools\)\.*/d'  
${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-oobe-  
desktop-ubuntu-20.04-rfs.MANIFEST.json
```

7. Update \${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-oobe-desktop-ubuntu-20.04-rfs.MANIFEST.json to use the DRIVE OS Core CopyTarget YAML manifests. You may update the “CopyTarget” key in the Build-FS manifest manually to include the YAML manifests from \${NV_WORKSPACE}/drive-linux/filesystem/copytarget/manifest/*.yaml or use the following single command (note that this is one single command that can be pasted into your bash terminal):

```
python3 -B - << END
import json
from collections import OrderedDict
bkConfig = OrderedDict()
manifest="${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-oobe-
desktop-ubuntu-20.04-rfs.MANIFEST.json"
with open(manifest) as f:
    data = f.read()
bkConfig = json.loads(data, object_pairs_hook=OrderedDict)
bkConfig["CopyTargets"] = [
    "\${COPYTARGETYAML_DIR}/copytarget-configs.yaml",
    "\${COPYTARGETYAML_DIR}/copytarget-aurix.yaml",
    "\${COPYTARGETYAML_DIR}/copytarget-aurix.yaml",
    "\${COPYTARGETYAML_DIR}/copytarget-firmware.yaml",
    "\${COPYTARGETYAML_DIR}/copytarget-headers.yaml",
    "\${COPYTARGETYAML_DIR}/copytarget-kernel-modules.yaml",
    "\${COPYTARGETYAML_DIR}/copytarget-libraries.yaml",
    "\${COPYTARGETYAML_DIR}/copytarget-samples.yaml",
    "\${COPYTARGETYAML_DIR}/copytarget-security.yaml",
    "\${COPYTARGETYAML_DIR}/copytarget-tools.yaml"
]
with open(manifest, 'w') as f:
    f.write(json.dumps(bkConfig, indent=4, sort_keys=False))
END
```

8. Build the final filesystem starting from ubuntu-base:

```
$ sudo -E /opt/nvidia/driveos/common/filesystems/build-fs/17/bin/build_fs.py -w
${NV_WORKSPACE}/ -i ${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-
oobe-desktop-ubuntu-20.04-rfs.MANIFEST.json -o $PWD/output/
```

7.4.9 File System Source Code

The filesystem source code packaged in nv-driveos-linux-ubuntu-20.04-src_<release>-<DRIVEOS_GCID>.deb, where <DRIVEOS_GCID> is the GCID of NVIDIA DRIVE OS Linux and <release> is the NVIDIA DRIVE OS Linux SDK version. The sources are built using the Debian package name and version from the manifest files in the rootfs below:

- > /etc/nvidia/rootfilesystem-manifest/driveos-core-ubuntu-20.04-rfs.MANIFEST.json
- > /etc/nvidia/rootfilesystem-manifest/driveos-oobe-ubuntu-20.04-rfs.MANIFEST.json

> /etc/nvidia/rootfilesystem-manifest/driveos-oobe-desktop-ubuntu-20.04-rfs.MANIFEST.json

The source is generated by executing the following command for each package (in the respective filesystem manifest) to obtain its source:

```
$ apt-get source <package>=<package_version>
```

The union of all the filesystem sources are delivered in the SDK.

NVIDIA DRIVE OS Linux filesystems contain a combination of Ubuntu repository and NVIDIA CUDA repository packages. As an alternative to above source, you can get the source of Ubuntu repository packages (from the Ubuntu repository) using the command:

Disclaimer:

Ensure that all data is saved. Execute a software shutdown command, such as halt/shutdown to the target system to avoid data corruption; otherwise, file system corruption may occur. Once the target system is shut down, you may use physical/electrical shutdown or reset commands, such as tegrareset or aurixreset in the AURIX command terminal.

7.4.10 VNC

DRIVE OS LINUX filesystems support using VNC access to DRIVE platform using canonical open source x11vnc. The solution supports both cases where a physical display is connected to the DRIVE platform (i.e. non-headless) or without any physical display (i.e. headless). The steps to use VNC broadly has 3 phases. The first phase sets the VNC mode (between headless vs non-headless), the second starts the X11 server (depending on the filesystems), and finally starts the x11vnc server on the DRIVE platform.

Step One: Selecting VNC mode between non-headless vs headless mode

What is the non-headless mode?

The non-headless mode runs X on the physical display connected to the DRIVE platform. It uses an accelerated nvidia driver stack and is the default mode.

The following shows xrandr output in non-headless mode:

```
Screen 0: minimum 8 x 8, current 1920 x 1080, maximum 32767 x 32767
DP-0 connected primary 1920x1080+0+0 (normal left inverted right x axis y axis) 477mm x
268mm
    1920x1080      60.00*+
    1680x1050      59.95
    1440x900       59.89
    1280x1024      75.02      60.02
```

Headless Mode

Headless mode is the mode where no physical display is connected to the target and VNC uses a virtual display over the network to connect/work with the target.

The following shows xrandr output in non-headless mode:

```
xrandr: Failed to get size of gamma for output default
Screen 0: minimum 320 x 240, current 1920 x 1080, maximum 1920 x 1080
default connected 1920x1080+0+0 0mm x 0mm
  1920x1080      60.00*
  1680x1050      70.00      60.00
  1400x1050      70.00      60.00
  1600x900       60.00
  1280x1024      75.00      60.00
```

Steps to set non-headless mode

The non-headless mode is the default mode in the filesystem and there are no actions required.

Steps to set headless mode

1. Backup current xorg.conf to xorg.conf.nvidia
 - a. sudo cp /etc/X11/xorg.conf /etc/X11/xorg.conf.nvidia
 - b. Note: To restore to non-headless copy /etc/X11/xorg.conf.nvidia over to /etc/X11/xorg.conf.
 - i. sudo cp /etc/X11/xorg.conf.nvidia /etc/X11/xorg.conf
2. Copy below xorg.conf content below to /etc/X11/xorg.conf.dummy. Copy /etc/X11/xorg.conf.dummy to /etc/X11/xorg.conf.
 - a. sudo cp /etc/X11/xorg.conf.dummy /etc/X11/xorg.conf

```
Section "Monitor"
  Identifier "Monitor0"
  HorizSync 28.0-80.0
  VertRefresh 48.0-75.0
  # https://arachnoid.com/modelines/
  # 1920x1080 @ 60.00 Hz (GTF) hsync: 67.08 kHz; pclk: 172.80 MHz
  Modeline "1920x1080_60.00" 172.80 1920 2040 2248 2576 1080 1081 1084 1118 -HSync
+Vsync
EndSection
Section "Device"
  Identifier "Card0"
  Driver "dummy"
  VideoRam 256000
EndSection
Section "Screen"
  DefaultDepth 24
  Identifier "Screen0"
  Device "Card0"
  Monitor "Monitor0"
  SubSection "Display"
    Depth 24
    Modes "1920x1080_60.00"
  EndSubSection
EndSection
```

Step Two: Starting X Server on the Filesystem

Desktop FS

For the oobe-desktop filesystem, the X server is automatically launched by default by the gdm3 display manager. No actions are required.

OOBE FS and CORE FS

For oobe/core filesystems, the X server is not launched on boot and requires manually starting it:

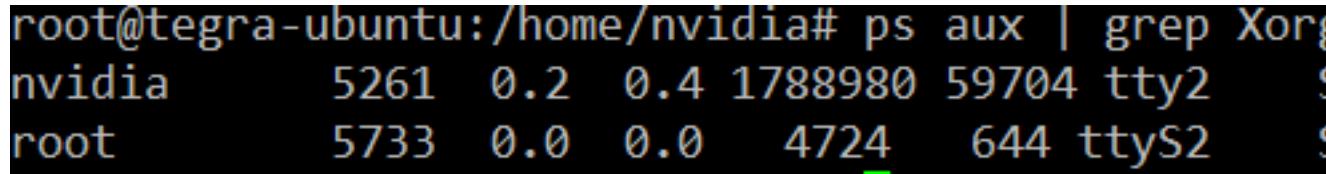
```
sudo -b X -ac -noreset -nolisten tcp
```

For more information, see [To start the X server](#).

Step Three: Running VNC server on DRIVE OS Filesystem and connecting to it

With Xserver running, follow the steps below to set up the x11vnc server and connect to it from the host VNC client.

Desktop FS

1. The gdm3 service automatically starts on desktop FS loading.
2. With x11vnc installed, please start x11vnc with cmdline args: (sudo is required because we are accessing another user's Xauthority)
 - a. `sudo DISPLAY=:0 x11vnc -auth /run/user/110/gdm/Xauthority -forever -noxdamage -repeat -shared -loop`
3. From the host side, the VNC client can connect to x11vnc (at port 5900) and see the greeter screen.
4. Note that after providing username & password in greeter, you will reach a blank screen because the GUI desktop gets started in a different X server instance.
5. To find the Xserver instance and Xauthority file, please use the ps command as shown below and note the Xauthority file <Xauth>:
 - a. `ps aux | grep Xorg | grep -v 110`
 - b. Look for the argument of the -auth option of the running Xorg
 - c. 

```
root@tegra-ubuntu:/home/nvidia# ps aux | grep Xorg
nvidia      5261  0.2  0.4 1788980 59704 tty2      S
root       5733  0.0  0.0   4724     644 ttys2      S
```
- d. In the above example: the Xauthority file is the argument to `-auth /run/user/1000/gdm/Xauthority`. So, the Xauthority file is `/run/user/1000/gdm/Xauthority`.
6. Now connect x11vnc using Xauthority file from step-5 to see the desktop:
 - a. `DISPLAY=:1 x11vnc -auth <Xauthority file> -forever -noxdamage -repeat -shared -loop`
 - b. As per example in step-5c, example cmdline is `DISPLAY=:1 x11vnc -auth /run/user/1000/gdm/Xauthority -forever -noxdamage -repeat -shared -loop`

OOBE FS and CORE FS

1. With x11vnc installed, please start x11vnc with cmdline args:
 - a. x11vnc -forever -noxdamage -repeat -shared -loop &
2. Connect to x11vnc from the host via vncviewer, to see a black screen because no applications are running on the screen.
3. Open a new terminal (for example ssh connection) and execute the following commands to run sample graphics app bubble:
 - a. cd /opt/nvidia/drive-linux/samples/opengles2/bubble
 - b. DISPLAY=:0 ./x11/bubble

7.4.11 Read-Only File System Considerations

DRIVE OS Linux file systems are supported to be mounted read-only when required by users. When DRIVE OS Linux Initramfs sees file systems that are mounted read-only, it enables access to a scratch partition gos0-rw-overlay of size 1 GB to receive the writes on the file system paths: /var, /tmp, /home, /etc. This scratch partition is not persistent across flashing and gets wiped during flashing.

For more information, see [DM-Verity and Read-Only File System Support](#).



Note: The read-only file system does not affect the support for persistent partitions. The mounts for persistent partitions for user metadata and user data are stacked on top of the gos0-rw-overlay partition. Therefore, writes /etc/ and /home directly go to their respective persistent partitions instead of the gos0-rw_overlay partition.

7.5 DRIVE OS Linux Filesystems Customization Quick Start Page

DRIVE OS Linux includes three filesystems, driveos-core-rfs, driveos-oobe-rfs, and driveos-oobe-desktop-rfs. This section goes over the basics of the LINUX filesystem customization tools and key examples.

7.5.1 Getting Started

To help you get started, this section provides two simple examples:

1. [How to use the filesystem tool Build-FS to rebuild the SDK Linux filesystem without any modifications](#)
2. [How to add a single debian package and a single file into your customized Linux filesystem](#)

Prerequisites

Ensure that the following tools are installed:

- > Build-FS
- > CopyTarget

See [Installing Build-FS](#) tool for instructions on installing and setting up these tools.

7.5.2 Rebuilding the Linux SDK File System Without Modifications Using the Build-FS Tool

This example shows the steps to rebuild the Linux SDK file system using the file system tool Build-FS. At the completion of this example, you will complete the following tasks:

- > Recreate the Linux file system without any modifications.
- > Flash your recreated Linux file system image onto the platform.
- > Boot up the target platform with your recreated Linux file system.

1. Run the build-fs tool using the following commands to create a file system without any changes. The created file system shall be at path \${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/nvidia-driveos-build-fs-user-rfs.img because of the output name specified in /opt/nvidia/driveos/common/filesystems/build-fs/17/configs/driveos-user-rfs.CONFIG.json. Copy the rebuilt file system to replace the starting file system at \${NV_WORKSPACE}/drive-linux/filesystem/targetfs.img.

```
export PROD_SUFFIX=""
export NVRTKERNELNAME=$(basename ${NV_WORKSPACE}/drive-linux/kernel/preempt_rt
${PROD_SUFFIX}/modules/*rt*-tegra)"
sudo -E /usr/bin/python3 -B /opt/nvidia/driveos/common/filesystems/build-fs/17/bin/
build_fs.py -w ${NV_WORKSPACE} -i /opt/nvidia/driveos/common/filesystems/build-
fs/17/configs/driveos-user-rfs.CONFIG.json -o ${NV_WORKSPACE}/drive-linux/filesystem/
targetfs-images/
sudo rm -f ${NV_WORKSPACE}/drive-linux/filesystem/targetfs.img
sudo ln -s ${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/nvidia-driveos-
build-fs-rfs-user.img ${NV_WORKSPACE}/drive-linux/filesystem/targetfs
```

2. The NVIDIA DRIVE OS flashing tool bootburn.py PCT picks up file system by default at path \${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/targetfs.img.
3. Use bootburn.py to flash the file system.
4. Reboot the target.
5. The target should have rebooted with your recreated Linux file system.

After step 5, the target boots the system and reaches the command prompt as shown below:

```
tegra-ubuntu login:
```

Enter the <username>/<password> corresponding to the user account(s) created during the initial setup of the NVIDIA DRIVE platform.

```
tegra-ubuntu login: <username>
password: <password>
```

After successful login, the console welcome prompt is seen:

```
<username>@tegra-ubuntu:~$
```

7.5.3 Adding a Single Debian Package and a Single File to the Linux File System

This example shows the steps to add a file, a Debian package, and build the Linux file system using the file system tools Build-FS and CopyTarget. At the completion of this example, you will complete the following tasks:

- > Customize your Linux file system.
 - > Flash your customized Linux file system image onto the NVIDIA DRIVE[®] platform.
 - > Boot up the target platform with your customized Linux file system.
 - > Verify the added package and the added file on the target.
1. In this example, we have chosen to use the file \${NV_WORKSPACE}/drive-linux/filesystem/content/samples/hello_world.txt that is part of the NVIDIA DRIVE OS samples to add to the Linux file system.
 2. You are, otherwise, welcome to use any other file you want, but the rest of this example will refer to hello_world.txt.
 3. Create the CopyTarget Manifest \${NV_WORKSPACE}/drive-linux/filesystem/copytarget/manifest/copytarget-hello.yaml and add entry to \${NV_WORKSPACE}/drive-linux/filesystem/content/samples/hello_world.txt as shown below. A list of file metadata (perm, owner, group) must be specified to provide detail to describe the file.
 4. To customize the standard Linux file system, specify the file system variant that the file hello_world.txt is part of. Importantly, you must provide the source location (pdk_sdk_installed_path) and the destination that the file hello_world.txt should reside in the customized Linux file system.
 5. As per steps 2 and 3, an example that describes hello_world.txt is provided in the code block below. Copy this excerpt into \${NV_WORKSPACE}/drive-linux/filesystem/copytarget/manifest/copytarget-hello.yaml.

```
fileList:
  - destination: /hello_world.txt
    source:
      pdk_sdk_installed_path: ${NV_WORKSPACE}/drive-linux/filesystem/content/
samples/hello_world.txt
      perm: 644
      owner: root
      group: root
```

6. Further, in this example, we demonstrate adding a Linux Debian package to the file system using the Build-FS config. To add a package, an entry must be added to the DebianPackages block of the config file.
7. Update the /opt/nvidia/driveos/common/filesystems/build-fs/17/configs/driveos-example-hello_world-rfs.CONFIG.json Build-FS config to add the CopyTarget YAML created above in the "CopyTargets" block and the *parted* Debian package in the DebianPackages block of the config file.

```
{
```

```

    "OS": "linux",
    "Output": "driveos-example-hello_world-rfs",
    "Base": "${BASE_DIR}/targetfs.img",
    "Mirrors": [
        {
            "Type": "local_debian_folder",
            "Path": "${MIRROR_DIR}"
        }
    ],
    "CopyTargets": [
        "${COPYTARGETYAML_DIR}/copytarget-hello.yaml"
    ],
    "DebianPackages": [
        "bsdmainutils"
    ]
}

```

8. Run the build-fs tool using the following commands to create a file system with preceding customizations. The created file system shall be at path \${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-example-hello_world-rfs.img because of the output name specified in /opt/nvidia/driveos/common/filesystems/build-fs/17/configs/driveos-example-hello_world-rfs.CONFIG.json. Copy the rebuilt file system to replace the starting file system at \${NV_WORKSPACE}/drive-linux/filesystem/targetfs.img.

```

export PROD_SUFFIX=""
export NVRTKERNELNAME=$(basename ${NV_WORKSPACE}/drive-linux/kernel/preempt_rt
${PROD_SUFFIX}/modules/*rt*-tegra)"
sudo -E /usr/bin/python3 -B /opt/nvidia/driveos/common/filesystems/build-fs/17/bin/
build_fs.py -w ${NV_WORKSPACE} -i /opt/nvidia/driveos/common/filesystems/build-
fs/17/configs/driveos-linux-example-hello_world-rfs.CONFIG.json -o ${NV_WORKSPACE}/
drive-linux/filesystem/targetfs-images/
sudo rm -f ${NV_WORKSPACE}/drive-linux/filesystem/targetfs.img
sudo ln -s ${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/driveos-example-
hello_world-rfs.img ${NV_WORKSPACE}/drive-linux/filesystem/targetfs.img

```

9. The NVIDIA DRIVE OS flashing tool bootburn.py PCT picks up the file system by default at path \${NV_WORKSPACE}/drive-linux/filesystem/targetfs.img.
10. Use bootburn.py to flash the file system.
11. Reboot the target.
12. The target should have rebooted with your customized Linux file system.
13. Verify that intended customizations are present:
 - a. Check ls at / path to verify /hello-world.txt exists:
 - i. \$ ls /hello-world.txt
 - b. Use the following hexdump command to confirm the bsdmainutils package is installed:
 - i. \$ hexdump -C /hello-world.txt

7.6 CopyTarget

The CopyTarget tool copies files reliably from a source location to the destination location while maintaining file metadata, which includes ownership and permission.

CopyTarget supports the following operations:

- > Copy file
- > Create directory
- > Update file metadata
- > Update directory metadata
- > Remove file
- > Remove empty directory
- > Create symlink

Prerequisites

- > Ubuntu 18.04 x86_64 host
- > python3 (>= 3.5)
- > python3-yaml
- > coreutils
- > bash

These are required to run the CopyTarget tool.

7.6.1 Installing CopyTarget

The CopyTarget tool is packaged and distributed as part of DRIVE OS SDK and can be installed by following the instructions in [Installation Using DRIVE OS Local Debian Packages](#).

CopyTarget is installed on the host at:

```
/opt/nvidia/driveos/common/filesystems/copytarget/1.4/
```

7.6.2 Using CopyTarget

- > Execute CopyTarget:

```
/opt/nvidia/driveos/common/filesystems/copytarget/1.4.5-<DRIVEOS_RELEASE>/copytarget.py
${TARGET_DIRECTORY} ${WORKSPACE} MANIFEST.yaml
```

Where:

- > <DRIVEOS_RELEASE> is the release version of NVIDIA DRIVE OS.
- > \${TARGET_DIRECTORY} is the destination directory.
- > \${WORKSPACE} is the workspace directory.

- > **MANIFEST.yaml** is the manifest file that specifies the file operations to execute.

7.6.2.1 Command Line Syntax

CopyTarget has the following invocation syntax with the following mandatory and optional arguments. For more information, see [Command Argument Options](#).

```
copytarget.py ${TARGET_DIRECTORY} ${WORKSPACE}
MANIFEST.yaml [MANIFEST1.yaml...] [OPTIONS]
```

The destination \${TARGET_DIRECTORY}, current workspace \${WORKSPACE}, and an initial manifest must be specified. Optional and additional manifests can be specified and are treated as if the initial manifest imported these additional manifests in its import section.

See [Command Argument Options](#) for additional options that can be specified to CopyTarget as part of [OPTIONS].

7.6.3 Command Argument Options

Argument	Description
--autocreate-parent-directories=AUTOCREA	Automatically create parent directories even if they have not been explicitly specified in the manifest. Accepted values: 'True/yes', 'False/no'. Default: 'True'
--buildfile-header=QNXBUILDFILEH	Add an header specified using this argument to QNX BuildFile
--blacklist=BLACKLISTY	Specify the path to a YAML manifest with a list of source path of files that should not be copied to TARGET_DIRECTORY. This is an operation to verify the CopyTarget YAML Manifest

Argument	Description
--buildfile-header-file=QNXBFHEADERFILE	Source file to include when generating buildfiles. Can be specified multiple times.
--version, -v	Prints the program's version number and exits.
--create-buildfile=QNXBUILDIFI	Convert YAML copytarget manifests to QNX BuildFile.
--help, -h	Show this help message and exits.
-u UIDMAPFILE, --user-identifier-dictionary=UIDMAPFILE	Explicitly specify the path to the passwd file. This is useful if the target root path does not have \${TARGET_DIRECTORY}/etc/passwd file.
-g GIDMAPFILE, --group-identifier-dictionary=GIDMAPFILE	Explicitly specify the path to the group file. This is useful if the target root path does not have \${TARGET_DIRECTORY}/etc/group file.
-s SOURCETYPE, --source-type=SOURCETYPE	Specify source type. Default value: "pdk_sdk_installed_path".
-f FILESYSTEMTYPE, --filesystem-type=FILESYSTEMTYPE	Specify filesystem type. Only process items in fileList that have required set to yes under filesystems entry in manifest. Default value: "standard".

Argument	Description
--no-chown	Do not change ownership of the files.
--whitelist=WHITELISTY	Specify the path to a YAML manifest with a list of source path of files that can be copied to TARGET_DIRECTORY. This is an operation to verify the CopyTarget YAML Manifest.
--verify-only	Only verify the CopyTarget YAML manifest. By specifying this option CopyTarget will only verify the CopyTarget manifest. At least one of the following arguments are necessary to specify what needs to be verified: 1) Check for blacklisted files with --blacklist. 2) Check for whitelisted files with --whitelist.

7.6.4 CopyTarget Manifest

CopyTarget operates on one or more manifest files and at least one must be provided to the tool. Each manifest file contains the source and destination locations of the file, as well as the metadata. Manifests follow YAML syntax, a human-readable data-serialization language.

A manifest is specified by two sections: a header, and a file list. The header and file list serve different roles. In the next section, the roles of these sections are explained in detail.

7.6.4.1 Header

The header contains information required to process the file list section of the manifest. The header also provides information indicating the compatible CopyTarget software. The header includes the following information as captured and detailed in the table.

Attribute	Description	Required	Default	Domain	Example
version	Defines the list of CopyTarget versions that the manifest is compatible with.	Mandatory	-	Decimal number	<code>version: '1.4'</code>
exports	Subsection containing a list of key value pairs. The keys defined in this section are used as variable identifiers and using these keys in the filelist expands them into assigned values.	Optional	Empty list	Key: a valid variable name Value: any valid YAML assignable value	exports: - Key1: Value1 - Key2: Value2
imports	Subsection containing a list of additional manifests.	Optional	Empty list	List of zero, one, or more valid and compatible filepaths to additional CopyTarget manifest files. Filepaths can be absolute or relative to the current manifest file; and this relativity is applied recursively for each nested manifest file. Imported manifest files are processed first before continued processing of the current manifest; however, exports from the current manifest are processed and forwarded to nested manifests. Defining	imports: - manifest1.yaml

7.6.4.2 Filelist

The filelist section defines an itemized list of files to be copied from source to destination. The list is processed in the order specified in the section. The filelist contains zero, one, or more file sections. Each file section includes the following information as captured and detailed in the table.

Attribute	Description	Required	Default	Domain	Example(s)
destination	<p>Specifies the path of the file, directory, or symlink that needs to be copied to or created in the target directory.</p> <p>The name of the destination file does not need to match the filename specified in the source attribute; when there is a difference, the file will be renamed accordingly.</p> <p>The value specified in destination attribute is relative to \${TARGET_DIRECTORY}, which must be specified to CopyTarget.</p>	Mandatory	-	Valid Linux filepath.	destination: '/lib/nvidia/lib_nvidia.so'

Attribute	Description	Required	Default	Domain	Example(s)
source	<p>Specifies the source path of a file or symlink that needs to be copied from.</p> <p>Sources must be specified using a dictionary data type. The key defines the source type and the corresponding value specifies the path. While executing copytarget, the key to be selected is determined by the --source-type parameter.</p> <p>This attribute should not be set when creating a directory or changing the permission of an already existing file or directory.</p> <p>The value specified in source attribute is relative to \${WORKSPACE}, which must be specified to CopyTarget (except when creating symlinks).</p>	Optional	""	Dictionary Key: Sting Value: Valid Linux filepath.	source: pdk_sdk_installed_path: /drive/libnvidia.so build_tree_path: /lib/libnvidia.so
perm	Specifies the permission of the file or directory in octal notation.	Mandatory (exception: creating symlinks)	-	Valid Octal notation of Linux permission.	perm: 0777

Attribute	Description	Required	Default	Domain	Example(s)
owner	<p>Specifies the owner identifier of the file, directory, or symlink and can be assigned a numeric UID or a string specifying the username.</p> <p>Strings are converted to numeric UID when a User Identifier Dictionary is provided to CopyTarget, see Identifier Dictionary section.</p>	Mandatory	-	Valid Linux UID, or UID alias.	<code>owner: root</code> <code>owner: 0</code>
group	<p>Specifies the group identifier of the file, directory, or symlink and can be assigned a numeric GID or a string specifying the groupname.</p> <p>Strings are converted to numeric GID when a Group Identifier Dictionary is provided to CopyTarget, see Identifier Dictionary section.</p>	Mandatory	-	Valid Linux GID, or GID alias.	<code>group: root</code> <code>group: 0</code>

Attribute	Description	Required	Default	Domain	Example(s)
filesystems	<p>Specifies the filesystem types that a file can be copied into.</p> <p>The key specifies the name of filesystem. The value field is a dictionary that defines whether a file can be copied into this filesystem by setting required:yes or required:no. The value field can also define additional key:values for purposes of bookkeeping. The additional fields are however not processed by CopyTarget.</p> <p>If this field is not defined, the file item is processed for ALL filesystem-types.</p>	Optional	ALL filesystem types	Dictionary Key: Sting Value: Dictionary	filesystems: standard: required: no safety: required: yes jama: XYZ
create_symlink	<p>If set to true, a symlink is created at the destination location. The symlink points to the value specified in the attribute source.</p> <p>By default, this field is set to false.</p>	Optional	false	true, false	<code>create_symlink : true</code>
remove	<p>If set to true, the file specified in the destination attribute is removed.</p> <p>If the destination attribute points to a directory, the directory must be empty; otherwise, CopyTarget reports an error; this is intentional and prevents inappropriate deletion.</p> <p>By default, this field is set to false.</p>	Optional	false	true, false	<code>remove: true</code>

7.6.5 CopyTarget File Operations

CopyTarget supports a limited number of file operations and to have CopyTarget execute a particular operation, a combination of file attributes must be specified. Any invalid combination is raised as an error by CopyTarget and an error message with a pointer to the offending file attribute is presented to the user to resolve.

The following sections describe each of the valid operations and the required attributes to trigger the operation.

Copy

Copy file from \${WORKSPACE}/libraries/lib_nvidia.so to \${TARGET_DIRECTORY}/usr/lib/lib_nvidia.so with ownership root:root and permission 0777; where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget, pdk_sdk_installed_path is the source type, and \${WORKSPACE} is the workspace directory provided to CopyTarget.

Attribute	Requirement	Example
destination	Required.	/usr/lib/lib_nvidia.so
source	Required (unless an imported manifest already defines it).	pdk_sdk_installed_path:libraries/lib_nvidia.so
perm	Required (unless an imported manifest already defines it).	0777
owner	Required (unless an imported manifest already defines it).	root
group	Required (unless an imported manifest already defines it).	root
create_symlink	Can be absent (default is false) or must be set to false.	false
remove	Can be absent (default is false) or must be set to false.	false

Create Directory

Create directory at \${TARGET_DIRECTORY} / usr /lib/ nvidia with ownership root:root and permission 0777; where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.

Attribute	Requirement	Examples
destination	Required (must end with a trailing slash; otherwise, an error is reported).	/usr/lib/nvidia/
source	Must be absent.	
perm	Required (unless an imported manifest already defines it).	0777
owner	Required (unless an imported manifest already defines it).	root
group	Required (unless an imported manifest already defines it).	root
create_symlink	Can be absent (default is false) or must be set to false.	false
remove	Can be absent (default is false) or must be set to false.	false

Update File Metadata

Set file \${TARGET_DIRECTORY}/usr/lib/lib_nvidia.so with ownership nvidia:nvidia and permission 0644; where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.

Attribute	Requirement	Example
destination	Required.	/usr/lib/lib_nvidia.so
source	Must be absent.	
perm	Required (unless an imported manifest already defines it).	0644
owner	Required (unless an imported manifest already defines it).	nvidia

Attribute	Requirement	Example
group	Required (unless an imported manifest already defines it).	nvidia
create_symlink	Can be absent (default is false) or must be set to false.	false
remove	Can be absent (default is false) or must be set to false.	false

Update Directory Metadata

Set directory \${TARGET_DIRECTORY}/usr/lib/nvidia with ownership nvidia:nvidia and permission 0644; where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.



Note: This operation is not recursive and only updates the metadata of \${TARGET_DIRECTORY}/usr/lib/nvidia/.

Attribute	Requirement	Example
destination	Required.	/usr/lib/nvidia/
source	Must be absent.	
perm	Required (unless an imported manifest already defines it).	0644
owner	Required (unless an imported manifest already defines it).	nvidia
group	Required (unless an imported manifest already defines it).	nvidia
create_symlink	Can be absent (default is false) or must be set to false.	false
remove	Can be absent (default is false) or must be set to false.	false

Remove a File

Remove file \${TARGET_DIRECTORY}/usr/lib/lib_nvidia.so; where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.

Attribute	Requirement	Example
destination	Required.	/usr/lib/lib_nvidia.so
source	Should be absent.	-
perm	Should be absent (if specified, is ignored).	-
owner	Should be absent (if specified, is ignored).	-
group	Should be absent (if specified, is ignored).	-
create_symlink	Should be absent (default is false) or must be set to false. Setting this to true causes an error to be thrown.	false
remove	Must be set to true. (unless an imported manifest already defines it).	true

Remove Empty Directory

Remove empty directory \${TARGET_DIRECTORY}/ `usr /lib/ nvidia /`; where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.

Attribute	Requirement	Example
destination	Required.	/usr/lib/nvidia/
source	Should be absent.	-
perm	Should be absent (if specified, is ignored).	-
owner	Should be absent (if specified, is ignored).	-
group	Should be absent (if specified, is ignored).	-
create_symlink	Must be set to false (unless an imported manifest already defines it). Setting this to true causes an error to be thrown.	false
remove	Must be set to true (unless an imported manifest already defines it).	true

Create Symlink

Create symlink at \${TARGET_DIRECTORY}/usr/lib/lib_nvidia.so.1 pointing to /usr/lib/lib_nvidia.so; where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.

Attribute	Requirement	Example
destination	Required.	/usr/lib/lib_nvidia.so.1
source	Required (unless an imported manifest already defines it).	pdk_sdk_installed_path: /usr/lib/lib_nvidia.so
perm	Should be absent (if specified, is ignored).	-
owner	Required (unless an imported manifest already defines it).	nvidia
group	Required (unless an imported manifest already defines it).	nvidia
create_symlink	Must be set to true (unless an imported manifest already defines it).	true
remove	Must be set to false (unless an imported manifest already defines it). Setting this to true causes an error to be thrown.	false

7.6.6 Identifier Dictionary

An identifier dictionary must be provided to CopyTarget if username alias and/or groupname alias are to be translated to UID and GID, respectively. Alias is not allowed in CopyTarget file operations due to the potential mismatches between systems. However, alias is permitted in CopyTarget manifests but are translated to numeric identifiers prior to file operation. In order to support translation, username identifier and/or groupname identifier mappings are required.

It is not required to supply both user identifier and group identifier dictionaries; especially, if alias are only used in one of the two categories. If alias is used in both categories, then it is required to provide CopyTarget with both identifier dictionaries. If one or more alias is specified in the manifest but no corresponding identifier dictionary is provided, CopyTarget will throw an error. If an identifier dictionary is provided but the alias is not found within, an error will also be thrown.

Valid identifier dictionaries include any valid password or group files; these are normally, /etc/passwd, and /etc/group, respectively. Any other formatted files are uninterpretable, and an error will be thrown.

7.6.6.1 Default Identifier Dictionary

When no Identifier Dictionary is specified in the corresponding category (user, or group), CopyTarget will search for any existing Identifier Dictionary relative to the \${TARGET_DIRECTORY} that is specified by the user. The default search paths are \${TARGET_DIRECTORY}/etc/passwd and \${TARGET_DIRECTORY}/etc/group and if any one of these exist, CopyTarget will take the values from these dictionaries.

Occasionally, users may not wish to use the dictionaries residing in the \${TARGET_DIRECTORY}; however, if such dictionaries do exist, CopyTarget will attempt to use them. If the user does not want to use default search paths, the command arguments corresponding to the dictionaries must be provided and pointed to an empty string. This means that one or both argument parameters should be set to empty string as in --user_identifier_dictionary="" and/or --group_identifier_dictionary="".

7.6.6.2 Host Identifier Dictionary

When CopyTarget is invoked on a \${TARGET_DIRECTORY} intended for the host, the user has the option to use the default identifier dictionaries (/etc/passwd and /etc/group) when \${TARGET_DIRECTORY} is "/", the root most path. This is the ideal situation.

However, if \${TARGET_DIRECTORY} refers to any other subpath location, then the user must invoke CopyTarget with the arguments specifying the host's identifier dictionaries. This is required as the target directory does not make available the identifier dictionaries, unless of course, a different set of identifier dictionaries are to be used, or that the manifest does not contain any user or group name alias; then in this case, it is not required to invoke CopyTarget with such arguments.

The following is an example of executing CopyTarget on a target directory intended for the host but is not in the root most location.

```
/opt/nvidia/driveos/<DRIVEOS_RELEASE>/filesystems/
copytarget/copytarget.py /target_directory/ /workspace/
MANIFEST.yaml --user_identifier_dictionary=/etc/passwd
--group_identifier_dictionary=/etc/group
```

Where:

- <DRIVEOS_RELEASE> is the release version of NVIDIA DRIVE OS.

7.6.7 Importing Manifests

Zero, one, or more additional manifests can be specified in the parent manifest. Further nested manifests can also be recursively achieved; since child manifests can have additional manifest imports. This behavior allows for hierarchical overlaying.

7.6.7.1 Overriding Parent Attributes

Hierarchical manifests allow for child manifests to be overridden, additional attributes to parent to be added, or ancestor manifests.

For example, a given child manifest may define a file attribute to have a permission attribute of 0777 and a parent manifest can override this permission attribute at will with another value such as 0666.

7.6.7.2 Mapping Overlays using Destination Field

Overlaying of manifests require the use of the destination key. When the destination key matches that of one of the parent manifest's specified file attribute, any values specified in the particular child manifest overlays (overrides) the running set of updated values, which in turn can be updated by another intermediate parent manifest.

Therefore, for overlaying to operate effectively, destination keys must match. Any destination keys that do not match are treated as a new file attribute, with its own set of specified required attributes. One or more missing attributes result in an error.

7.6.7.3 Sequence of Manifest Overlaying

The sequence of manifest processing is dictated by the ordering specified in the import section of the parent manifest and the processing of this list follows a depth first list.

Consider the following set of manifest examples, which causes the final file attribute to have a resultant value depending on the ordering of the root most manifest (which is provided first to CopyTarget). See the result below.

```
#manifest0.yaml
imports:
  - manifest1a.yaml
  - manifest1b.yaml
fileList:
  - destination: /destination.txt
    source:
      pdk_sdk_installed_path: /source.txt
      perm: 644
      owner: 0
      group: 0
#manifest1a.yaml
imports:
  - manifest2a.yaml
  - manifest2b.yaml
fileList:
  - destination: /destination.txt
    source:
      pdk_sdk_installed_path: /source.txt
      perm: 644
      owner: 1000
      group: 1000
#manifest2a.yaml
```

```

fileList:
  - destination: /destination.txt
    source:
      pdk_sdk_installed_path: /source.txt
      perm: 655
      owner: 2000
      group: 1500
#manifest2b.yaml
fileList:
  - destination: /destination.txt
    source:
      pdk_sdk_installed_path: /source2.txt
      perm: 777
      group: 3000
#manifest1b.yaml
fileList:
  - destination: /destination2.txt
    source:
      pdk_sdk_installed_path: /source.txt
      perm: 644
      owner: 1000
      group: 1000

```

7.6.7.3.1 Resultant Table

Starting Manifest	Copied File(s)	Ownership	Permission
manifest0.yaml	1. ./source2.txt to /destination.txt 2. ./source.txt to /destination2.txt	1. 0:0 2. 1000:1000	1. 0644 2. 0644
manifest1a.yaml	1. ./source2.txt to /destination.txt	1. 1000:1000	1. 0644
manifest1b.yaml	1. ./source.txt to /destination2.txt	1. 1000:1000	1. 0644
manifest2a.yaml	1. ./source.txt to /destination.txt	1. 2000:1500	1. 0655
manifest2b.yaml	Error, because owner attribute is not defined.	-	-

7.6.8 Exports

The exports section can be used for variable dereferencing in the current and child manifests.

7.6.8.1 Syntax

The syntax follows a key value pair, delimited by a colon. This follows the standard YAML syntax format. More than one export can be specified. The key must not have any spaces.

For example, the following code statement assigns the value VALUE to the variable named KEY.

```
exports:
  - KEY: VALUE
```

7.6.8.2 Dereferencing a Variable

To dereference or otherwise obtain the value stored in a particular key of an export, enclose the name of the variable between opening and closing braces {} and prepended with a dollar sign \$.

For example, the variable named KEY can be dereferenced when specified as \${KEY}, as illustrated in the following example, which sets the attribute destination to VALUE.

```
exports:
  - KEY: VALUE
fileList:
  - destination: ${KEY}
```

7.6.8.3 Overriding Parent Assigned Values

Values assigned previously to a variable from a parent manifest can be overridden by any child manifest and thus taken forward.

Variable assignments are part of the preprocessing step, which means that all variables are assigned the final values prior to iteration of any fileList section. This is intended to prevent variable value aliasing problems.

For example, in the following scenario, the attribute source of destination dest is assigned the value FINAL3, as would the destination dest2. The source attribute of destination dest2 is never assigned the value FINAL2, even though it appears in the intermediate manifest import because of the nature of preprocessing requirement.

```
#manifest1c.yaml
exports:
  - KEY: FINAL1
imports:
  - manifest2c.yaml
  - manifest2d.yaml
fileList:
```

```

- destination: dest
  source:
    pdk_sdk_installed_path: ${KEY}
#manifest2c.yaml
exports:
  - KEY: FINAL2
fileList:
  - destination: dest
    source:
      pdk_sdk_installed_path: ${KEY}
  - destination: dest2
    source:
      pdk_sdk_installed_path: ${KEY}
#manifest2d.yaml
exports:
  - KEY: FINAL3
fileList:
  - destination: dest
    source:
      pdk_sdk_installed_path: ${KEY}

```

7.6.9 Examples: Creating CopyTarget Manifest

7.6.9.1 Copying a File

The following file attribute instructs CopyTarget to copy the file from \${WORKSPACE} / home/nvidia/file.txt to \${TARGET_DIRECTORY} / file.txt with permission 0644 and ownership nvidia:nvidia, where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget and where \${WORKSPACE} is the workspace directory provided to CopyTarget. Since filesystems field is not defined, this file is copied to ALL filesystem-types.

```

fileList:
  - destination: /file.txt
    source:
      pdk_sdk_installed_path: /home/nvidia/file.txt
    perm: 644
    owner: nvidia
    group: nvidia

```

7.6.9.2 Copying a File to Specific FileSystem Type

The following file attribute instructs CopyTarget to copy the file from \${WORKSPACE} / home/nvidia/file.txt to \${TARGET_DIRECTORY} / file.txt with permission 0644, and ownership nvidia:nvidia; where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget and where \${WORKSPACE} is the workspace directory provided to CopyTarget. Based on the filesystems field, this file is copied to only "safety" or "safety-debug" filesystem-type.

```

fileList:
  - destination: /file.txt

```

```

source:
  pdk_sdk_installed_path: /home/nvidia/file.txt
perm: 644
owner: nvidia
group: nvidia
filesystems:
  standard:
    required: no
  safety:
    required: yes
    jama: DRV5X-STAKEHLDREQPLCL3-2429
  safety_debug:
    required: yes
    jama: DRV5X-STAKEHLDREQPLCL3-2429

```

7.6.9.3 Creating a Directory

The following file attribute instructs CopyTarget to create the directory \${TARGET_DIRECTORY}/usr/bin/ with permission 0644 and ownership nvidia:nvidia, where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.


Note:

If the destination directory already exists, the metadata of the \${TARGET_DIRECTORY}/usr/bin/ is set to values specified, overriding any original values..

```

fileList:
  - destination: /usr/bin/
    perm: 644
    owner: nvidia
    group: nvidia

```

7.6.9.4 Updating a File's Metadata

The following file attribute instructs CopyTarget to set the file \${TARGET_DIRECTORY}/file.txt with permission 0644 and ownership nvidia:nvidia, where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.

```

fileList:
  - destination: /file.txt
    perm: 644
    owner: nvidia
    group: nvidia

```

7.6.9.5 Removing a File

The following file attribute instructs CopyTarget to remove the file \${TARGET_DIRECTORY}/file.txt, where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.

```
fileList:
```

```
- destination: /file.txt
remove: true
```

7.6.9.6 Creating a Symlink

The following file attribute instructs CopyTarget to create a symlink at \${TARGET_DIRECTORY}/link.txt that points to /file.txt, where \${TARGET_DIRECTORY} is the target directory provided to CopyTarget.

```
fileList:
- destination: /link.txt
source:
    pdk_sdk_installed_path: /file.txt
create_symlink: true
```

7.6.9.7 Comprehensive Example: copytarget-manifest.yaml

An example of a comprehensive copytarget-manifest.yaml looks like the following.

```
version: '1.4'
# Exports environment variables
exports:
- SAMPLES_HOME: /home/nvidia/drive-linux/samples/
- FILE_VERSION: 1
# Itemised file list
fileList:
    # The following attribute creates a directory
"${TARGET_DIRECTORY}/nvidia/"
        # with ownership 0:0 and permission 0755; where
${TARGET_DIRECTORY} is the target directory specified to CopyTarget.
    - destination: /nvidia/
        perm: 0755
        owner: 0
        group: 0
    filesystems:
        standard:
            required: yes
    safety:
        required: yes
        jama: DRV5X-STAKEHLDREQPLCL3-2429
    safety_debug:
        required: yes
        jama: DRV5X-STAKEHLDREQPLCL3-2429
    # The following attribute copies the file from
"${WORKSPACE}/home/drive/samples/nvidia_sample"
        # to "${TARGET_DIRECTORY}/nvidia/nvidia_sample"
with ownership 1000:1000 and permission 0777 when "pdk_sdk_installed_path" source-type
is chosen.
        # The file is only copied when the chosen filesystem-type
is either safety or safety-debug.
    - destination: /nvidia/nvidia_sample
        source:
            pdk_sdk_installed_path: nvidia_sample
```

```

        build_tree_path: nvidia_sample_internal
perm: 0777
owner: 1000
group: 1000
filesystems:
    standard:
        required: no
    safety:
        required: yes
        jama: DRV5X-STAKEHLDREQPLCL3-2429
    safety_debug:
        required: yes
        jama: DRV5X-STAKEHLDREQPLCL3-2429
# The following attribute creates a symlink at
"${TARGET_DIRECTORY}/nvidia_sample_1"
# pointing to "/nvidia/nvidia_sample"; where
${TARGET_DIRECTORY} is the target directory specified to CopyTarget.
# In this example, the symlink is only created for "standard" filesystem.
- destination: /nvidia_sample_${FILE_VERSION}
source:
    pdk_sdk_installed_path: /nvidia/nvidia_sample
    build_tree_path: /nvidia/nvidia_sample
    create_symlink: true
filesystems:
    standard:
        required: yes
        jama: DRV5X-STAKEHLDREQPLCL3-1234
    safety:
        required: no
    safety_debug:
        required: no
# The following attribute removes file
"${TARGET_DIRECTORY}/nvidia/nvidia_sample";
# where ${TARGET_DIRECTORY} is the target
directory specified to CopyTarget.
# Since filesystems field is not defined, this
item will be process for ALL filesystem-types.
- destination: ${targetdir}/nvidia/nvidia_sample
remove: true

```

7.6.10 Errors

To enable proper tracking of files and to prevent erroneous file copies, CopyTargetv1.4 enforces the following guidelines. Violations are reported with an error and the line of the offending code.

Fault	Description	Error
Files cannot be copied implicitly.	Copying directories is not permitted.	In case of violation, CopyTarget throws an error saying, for example, Error: CopyTarget does not allow directory to directory copy. Please itemize the files to copy.
Use of wildcards in file names is not permitted.	Wildcard expansion is not permitted.	In case of violation, CopyTarget throws an error saying, for example, FileNotFoundError: [Errno 2] No such file or directory: '/usr/bin/*'.
Directory path entries must end with a slash (/).	Directories must end in a trailing slash to differentiate from files.	In case of violation, CopyTarget throws an error saying Error: Directory entries must end with a trailing slash.
Metadata of each file must be defined.	All metadata must be specified.	In case of violation, CopyTarget throws an error saying, for example, Error: Expected key 'owner' is not defined.

7.6.11 DRIVE OS SDK Copytarget Manifests

DRIVE OS SDK ships the copytarget manifest files that contain an itemized list of files in the filesystem and their locations in the SDK. The listing of each manifest .yaml file is as follows:

Copytarget YAML file	Description of copy listings	driveos-core-rfs	driveos-oobe-rfs
copytarget-configs	DRIVE OS Linux configuration files, services, helper scripts, and library metadata.	YES	YES
copytarget-security	Overlay files to enforce security features on the filesystem.	YES	YES
copytarget-firmware	DRIVE OS Linux platform firmware images and related files from BSP.	YES	YES
copytarget-headers	DRIVE OS Linux graphics opengl, khronos, headers.	YES	YES
copytarget-kernel-modules	DRIVE OS Linux kernel modules: both realtime and standard variant.	YES	YES

Copytarget YAML file	Description of copy listings	driveos-core-rfs	driveos-oobe-rfs
copytarget-libraries	DRIVE OS Linux driver and middleware libraries.	YES	YES
copytarget-tools	DRIVE OS Linux tools and applications.	YES	YES
copytarget-dpx	DRIVE OS Linux platform-specific applications, middleware, and services.	YES	YES
copytarget-aurix	DRIVE OS AURIX firmware.	YES	YES
copytarget-samples	DRIVE OS Linux samples for graphics, multimedia, security, and system software modules.	NO	YES

7.7 NVIDIA Build-FS



Note: Version 17.1.5

NVIDIA Build-FS is a tool for creating and customizing Linux filesystems. It is a common filesystem interface tool for automotive programs.

7.7.1 Key Features

- Supports generating and customizing target filesystem image and tarball (uncompressed archive).
- Supports upstream Debian mirrors.
- Supports adding and updating users, groups, and memberships in the target filesystem.
- Outputs manifest files describing the filesystem created.

7.7.2 Prerequisites

Refer to Drive OS hardware and software requirements.

7.7.2.1 Package Dependencies

```
# sudo
# wget
# python3
```

```
# mount
# qemu-user-static
# binfmt-support
# coreutils
# tar
# bash
# e2fsprogs (>= 1.42.8-1ubuntu1)
# dpkg-dev
```

7.7.3 Installing NVIDIA Build-FS



Note:

Prior to installing NVIDIA Build-FS, ensure that NVIDIA DRIVE OS Linux SDK is installed on the system. A compatible version of NVIDIA DRIVE OS must be installed corresponding to the version of NVIDIA Build-FS to be installed.



Note:

Prior to installing NVIDIA Build-FS, ensure that NVIDIA CopyTarget is installed on the system. A compatible version of NVIDIA CopyTarget must be installed corresponding to the version of NVIDIA Build-FS to be installed. For installation instructions, see [Installing CopyTarget](#).

The NVIDIA Build-FS tool is automatically installed as a part of DRIVE OS installation at `/opt/nvidia/driveos/common/filesystems/build-fs/<version>/`. The following instructions specify manual installation steps if DRIVE OS installation is not performed.

1. Download `nv-driveos-common-build-fs-<NV_BUILD_FS_VERSION>_<NV_BUILD_FS_VERSION>.amd64.deb`.
2. Install Debian package `nv-driveos-common-build-fs-<NV_BUILD_FS_VERSION>_<NV_BUILD_FS_VERSION>.amd64.deb` using the apt package manager.
3. NVIDIA Build-FS is installed to `/opt/nvidia/driveos/common/filesystems/build-fs/<NV_BUILD_FS_VERSION>/`.

The following example is a walkthrough to install NVIDIA Build-FS from a BASH terminal.

```
# Change directory to the location where nv-driveos-common-build-
fs-<NV_BUILD_FS_VERSION>_<NV_BUILD_FS_VERSION>.amd64.deb has been downloaded to. In this
example,
the Debian has been downloaded to ~/Downloads/.
nvidia@nvidia:~$ cd ~/Downloads/
# Install NVIDIA Build-FS.
nvidia@nvidia:~/Downloads/$ sudo apt install ./nv-driveos-common-build-
fs-<NV_BUILD_FS_VERSION>_<NV_BUILD_FS_VERSION>.amd64.deb
# Verify NVIDIA Build-FS has been installed successfully using the
--version command argument. The version reported should match that which
has been installed; in this example, NVIDIA Build-FS version 9 has been installed.
nvidia@nvidia:~/Downloads/$ /opt/nvidia/driveos/common/filesystems/build-fs-
<NV_BUILD_FS_VERSION>/bin/build_fs.py --version
```

Build-FS Version: <NV_BUILD_FS_VERSION>

7.7.4 Editing NVIDIA Build-FS CONFIG

NVIDIA Build-FS does not come with any predefined CONFIG or MANIFEST. Any predefined MANIFEST and the associated user CONFIG are downloaded and installed as part of NVIDIA DRIVE OS.

NVIDIA Build-FS does come with a default basic CONFIG template you can build upon. The default CONFIG template has nothing defined and produces no result when executed with NVIDIA Build-FS but acts as a starting point for you to learn.

The default CONFIG template is located within the configs/ directory of the installation location of NVIDIA Build-FS, which is usually located at /opt/nvidia/driveos/common/filesystems/build-fs/<NV_BUILD_FS_VERSION>/configs/. Within the directory, you can find the default CONFIG template, driveos-user-rfs.CONFIG.json.

This example walkthrough edits the default CONFIG template; however, if you wish, you may use your own or one of the NVIDIA DRIVE OS manifests or configurations for customization. This simple example installs the Debian vim, but any other Debian can be used. Open a BASH terminal and issue the following commands:

```
# Edit default CONFIG template using an editor of your choice,
# we chose vim for simplicity.
nvidia@nvidia:~$ vim /opt/nvidia/driveos/common/filesystems/build-fs/
<NV_BUILD_FS_VERSION>/configs/driveos-user-rfs.CONFIG.json
# Once the file has been open for edit, update the section "DebianPackages"
# and add the entry "vim" as shown.
"DebianPackages":
[
    "vim"
]
# Also update the section Mirrors as shown below, with the Ubuntu mirror
# from which "vim" Debian package can be obtained.
"Mirrors":
[
    "deb http://ports.ubuntu.com/ubuntu-ports/ focal main universe restricted"
]
# Once the changes have been made, commit your changes by saving the file.
# If you wish to have other changes, refer to section "NVIDIA Build-FS CONFIG
Semantics" for additional details of NVIDIA Build-FS CONFIG semantics.
```

You have edited your first NVIDIA Build-FS CONFIG successfully. The following sections describe how to generate an image and flash your image to the target platform.

7.7.5 Executing NVIDIA Build-FS With the Updated CONFIG

You can invoke NVIDIA Build-FS to produce a filesystem image. Continuing from the example in the previous section, in the existing or newly opened BASH terminal, execute the following walkthrough commands:

```
sudo -E /opt/nvidia/driveos/common/filesystems/build-fs/<NV_BUILD_FS_VERSION>/bin/build_fs.py -w ~/driveos/ -i /opt/nvidia/driveos/common/filesystems/build-fs/<NV_BUILD_FS_VERSION>/configs/driveos-user-rfs.CONFIG.json -o output/
```

Note: In this example, NVIDIA DRIVE OS is installed to `~/driveos/`; if NVIDIA DRIVE OS is installed to another directory, ensure that NVIDIA Build-FS is pointed to the appropriate workspace location.

A filesystem image named `nvidia-driveos-build-fs-rfs-user.img` (the default name specified in the default CONFIG template, but you can change the name) is generated under the `output/` directory. The filesystem image can be deployed and used to flash onto the target platform.

7.7.6 Flashing the Customized Target File System

1. Update the symbolic link (symlink) of `drive-linux/filesystem/targetfs.img` to point to your customized target file system image.
2. Flash the target.

```
nvidia@nvidia:~$ sudo mkdir -p ~/driveos/drive-linux/filesystem/targetfs-images/
nvidia@nvidia:~$ sudo mv output/nvidia-driveos-build-fs-rfs-user.img ~/driveos/drive-
linux/filesystem/targetfs-images/
nvidia@nvidia:~$ sudo rm -f ~/driveos/drive-linux/filesystem/targetfs.img
nvidia@nvidia:~$ sudo ln -sf targetfs-images/nvidia-driveos-build-fs-rfs-user.img ~/-
driveos/drive-linux/targetfs.img
# Where ~/driveos/, is the DRIVE OS Installation Directory.
```

7.7.7 NVIDIA Build-FS Architecture

NVIDIA Build-FS tool is a Python program that takes in a json CONFIG file as input and outputs an ext4 filesystem image.

NVIDIA Build-FS parses the input CONFIG file to determine the target OS desired and initializes OS specific class objects to process the filesystem creation process.

The filesystem creation process contains four stages, which are:

- > Pre-build
- > Build
- > Post-build
- > Process-output

These stages are present for all supported operating systems, but the implementation varies depending on the OS.

OS	Linux
pre-build	<ol style="list-style-type: none"> 1. Setup binfmts for executing different arch binaries. 2. Extract base filesystem. 3. Update Mirrors in the target filesystem with values provided in the input CONFIG. 4. Update resolv.conf of target filesystem with host's resolv.conf. 5. Run preinstall scripts.
build	<ol style="list-style-type: none"> 1. Generate comprehensive list of Debian packages to be installed. 2. Install Debian packages using package manager (apt). 3. Create users/groups and update user memberships. 4. Edit Hostname of the generated filesystem image. 5. Execute CopyTarget scripts.
post-build	<ol style="list-style-type: none"> 1. Restore target filesystem mirrors to original in the target filesystem. 2. Restore target filesystem resolv.conf to original in target filesystem. 3. Run postinstall scripts. 4. Generate MANIFEST. 5. Copy MANIFEST into the target filesystem for reference
process-output	<p>Create chosen target filesystems outputs:</p> <ol style="list-style-type: none"> 1. Create ext4 filesystem image 2. Create tarball (compressed archive) of the filesystem.

All Build-FS helper scripts/temporary files used in the process of generating the filesystem image are stored under the /tmp/ directory of the target filesystem during its required lifetime and are deleted before the final target filesystem image is created.

7.7.8 Command Line Arguments

These options are available when you execute build_fs.py.



Note:

Defaults are used if options are not used in the NVIDIA Build-FS command line.

7.7.8.1 Tool Information Command Line Arguments

Short Option/Long Option

Description**-h/--help**

Show help message and exit.

-v/--version

Print version and exit.

7.7.8.2 Required Command Line Arguments

Short Option	-i JSON_PATH
Long Option	--input=JSON_PATH
Value Type	JSON_PATH must be a valid UNIX filepath or the value 'STDIN'
Description	Specifies <JSON_PATH> as the absolute path to NVIDIA Build-FS CONFIG file or 'STDIN' to inform build-fs that CONFIG file is coming from standard-input.
Short Option	-w
Long Option	--nv-workspace=NV_WORKSPACE
Value Type	NV_WORKSPACE must be a valid UNIX directory path
Description	Specifies NV_WORKSPACE as the absolute path to workspace location; this is the path to the location of NVIDIA DRIVE OS.

7.7.8.3 Optional Command Line Arguments

Short Option	-o OUTPUT_FOLDER
Long Option	--output=OUTPUT_FOLDER
Value Type	OUTPUT_FOLDER must be a valid UNIX directory path
Defaults	`\${PWD}` (Value of the present working directory)
Required	No, but recommended.
Description	Specifies <OUTPUT_FOLDER> as the absolute path to folder for output.

Short Option	N/A
Long Option	--create-tar=CREATE_TAR
Value Type	CREATE_TAR must be an option in list: [yes, no]
Defaults	No
Required	No
Description	Specifies to create filesystem tarball compressed as BunZip2.
Short Option	N/A
Long Option	--create-image=CREATE_IMAGE
Value Type	CREATE_IMAGE must be an option in list: [yes, no]
Defaults	Yes
Required	No
Description	Specifies to create ext4 filesystem image.
Short Option	N/A
Long Option	--copytarget-source-type=COPYTARGET_SOURCE_TYPE
Value Type	COPYTARGET_SOURCE_TYPE must be a string with printable characters
Defaults	pdk_sdk_installed_path
Required	No
Description	Specifies the type of source path in the copytarget manifest, from which files must be copied to the target filesystem directory. For more information, see CopyTarget.
Short Option	-f FILESYSTEM_WORK_FOLDER

Long Option	--filesystem-working-directory=FILESYSTEM_WORK_FOLDER
Value Type	FILESYSTEM_WORK_FOLDER must be a valid UNIX directory path
Defaults	None.
Required	No
Description	<p>Allows user to override the folder in which the target filesystem is extracted and worked on. If option is not provided, target filesystem is extracted to BUILD_KIT_TMP_DIR/targetfs.</p> <p>Where BUILD_KIT_TMP_DIR is the temporary work directory created by the tool for storing temporary files and it is deleted once the output filesystem image/tarball is generated.</p>
Short Option	N/A
Long Option	--log-level=LOG_LEVEL
Value Type	LOG_LEVEL must be an option in list: ['debug', 'info', 'warning', 'error', 'critical']
Defaults	info
Required	No
Description	<p>Allows you to choose the verbosity of the build_fs tool.</p> <ol style="list-style-type: none"> 1. critical: Shows only critical errors 2. error: Shows all errors in addition to messages above 3. warning: Shows all warnings in addition to messages above 4. info: Shows all info messages in addition to messages above 5. debug: Shows all debug information in addition to messages above

7.7.9 Environment Configuration

The /opt/nvidia/driveos/common/filesystems/build-fs/<NV_BUILD_FS_VERSION>/build-fs.config file controls the execution environment of NVIDIA Build-FS.

It is a JSON file with fields:

- > common

Variables defined in this field are always present in the tool's execution environment.

- > linux

Variables defined in this field are defined if NVIDIA Build-FS CONFIG has OS set to linux. For more information, see [CONFIG Semantics](#).

7.7.9.1 Special Environment Variables

The following tables show Environment variables that impacts Build-FS behavior. These variables can only be set via the environment configuration file. If these variables are defined in the shell environment before invoking Build-FS, they are unset by the program before reading the environment configuration file.

Variable	REQUIRED_VARIABLES
Value	Comma separated variable names
Default	None
Use case	Checks if the variables names specified as comma separated values are defined in the environment. Exits if they are not defined.
Variable	QEMU_PATH
Value	Path to qemu-aarch64-static.
Default	/usr/bin/
Use case	Specifies the location of qemu-aarch64-static binary for chrooting into the Linux filesystem.
Variable	BUILD_FS_DIR
Value	Path to NVIDIA Build-FS directory.
Default	/opt/nvidia/driveos/common/filesystems/build-fs/ <NV_BUILD_FS_VERSION>
Use case	Specifies NVIDIA Build-FS location.
Variable	COPYTARGET
Value	Path to CopyTarget.

Variable	REQUIRED_VARIABLES
Default	/opt/nvidia/driveos/common/filesystems/copytarget/1/ copytarget.py
Use case	Specifies the location of NVIDIA CopyTarget.
Variable	PYTHON3
Value	Path to Python3.
Default	/usr/bin/python3
Use case	Specifies the location of Python3 required for invoking copytarget.

7.7.9.2 Environment Variables Available in Pre and Post Install Scripts

The following table lists out available environment variables that can be used throughout NVIDIA Build-FS and at which specific location. If a particular variable is not available for use, an indicator is marked in this table.

Variable	Description	PreInstall Scripts	PostInstall Scripts
WORK_DIR	Assigned value of the path to NVIDIA Build-FS's workspace.	Available	Available
FILESYSTEM_WORK_DIR	Assigned value of the path to target filesystem directory located on host.	Available	Available
BUILD_FS_OUTDIR	Assigned value of the path to Build-FS's output directory.	Available	Available
NV_WORKSPACE	Assigned value of the path to NVIDIA DRIVE OS installation	Available	Available

7.7.10 CONFIG Semantics

This section details the semantics of the attributes available in NVIDIA Build-FS CONFIG. Due to JSON parsing, the last value for a duplicated attribute is the one taken; any previously declared value is overwritten.

7.7.10.1 Legend

NVIDIA Build-FS CONFIG is a valid JSON file and hence we can consider the file as a JSON Object. The terminologies used here are from JSON (https://en.wikipedia.org/wiki/JSON#Data_types_and_syntax).

Field	List of all the keys supported by NVIDIA Build-FS CONFIG.
Type	Type of Value the keys should have.
Acceptable Values	<ul style="list-style-type: none"> > Additional information on what the Value should be. > Key content -> If Value is of type object, then its keys should be a String and must adhere to the conditions specified here. > Value type -> If Value is of type object, then the object's value must be of the type specified here. > Value contents -> If Value is of type object, then the object's value must adhere to the conditions specified here. > Items type -> If Value is of type array, then the array's values must be of the type specified here. > Items contents -> If Value is of type array, then the array's values must adhere to the conditions specified here. > enum [a, b, c] -> Implies Value must be one of a, b or c. > Valid UNIX filename -> Valid String, which do not contain characters specified here: https://en.wikipedia.org/wiki/Filename#Reserved_characters_and_words <p>Valid UNIX filepath -> Valid String, which is a combination of 'Valid UNIX filename'(s) joined via /, which should point to an existing file.</p>
REQ	<p>This row can have the following values:</p> <ol style="list-style-type: none"> 1. Yes -> The Field Must be present in the CONFIG with a non-empty value. 2. No -> If the Field is not present in the CONFIG, the default shall be used, else specified value is used.

Default	<p>Specifies the Default value for 'Not Required' fields if they are absent from the CONFIG file.</p> <p>The row can have the following standard values apart from actual default values.</p> <ul style="list-style-type: none"> > N/A -> Cannot have a default value. (Required) > empty -> Doesn't have a default value. (Getter shall return None)
Syntax	Specifies the Syntax of usage of the field in the CONFIG file. Useful for a Visual understanding of Type and Acceptable Values .
Instructions	Additional information on how the fields are used by Build-FS and what behavior would it cause.
Examples	Example usage of the field in the CONFIG file.

7.7.10.2 Required CONFIG Fields

Field	OS
Type	String
Acceptable Values	enum[linux]
Req	Yes
Default	N/A
Instructions	Specifies "linux" for generating Linux target filesystem image.
Example(s)	"OS": "linux"
Field	Output
Type	String
Acceptable Values	Valid UNIX filename without extension.
Req	Yes

Default	N/A
Instructions	Specifies the prefix name of the different filesystem output files generated
Example(s)	"Output": "nvidia-driveos-build-fs-user-rfs"

7.7.10.3 Optional CONFIG Fields

Field	Base
Type	String
Acceptable Values	Valid UNIX dirpath or a Valid UNIX filepath.
Req	No
Default	None
Instructions	<p>Specifies the path of the target filesystem to be used as input and to be built on top of.</p> <p>The value to this tag must either be:</p> <ol style="list-style-type: none"> 1. A folder 2. A mountable image file with the extension .img in the filename 3. A bzip compressed tar file with the extension .tar.bz2 in the filename <p>The filename must have an extension of .img or .tar.bz2;</p> <p>Base is always used when building a new filesystem.</p> <p>Base can be omitted when updating an existing filesystem. If Base is not provided, -f option is mandatory.</p>
Example(s)	<pre>"Base": "/home/nvidia/driveos/nvidia-driveos-base-rfs/"</pre> <pre>"Base": "/home/nvidia/driveos/nvidia-driveos-base-rfs.img"</pre> <pre>"Base": "/home/nvidia/driveos/nvidia-driveos-base-rfs.tar.bz2"</pre>
Field	PreInstalls
Type	Object

Field	Base
Acceptable Values	<p>Key content: Valid UNIX filepath to a BASH script.</p> <p>Value type: String Value content: enum[host, target, target_copy]</p>
Req	No
Default	None
Instructions	<p>Specifies a list of zero, one, or more BASH scripts to be executed on the host or virtualized target during the Pre-Build stage.</p> <p>For each object element, the value of the element specifies whether the specific script is to be executed on the host or virtualized target. A value of:</p> <ul style="list-style-type: none"> "host" is the indicator to execute existing BASH script on the host. "target" is the indicator to execute existing BASH script on the virtualized target. "target_copy" is the indicator to copy existing BASH script on the host to /tmp / of the target filesystem. execute the script in /tmp / on the virtualized target. remove the script in /tmp / of the target filesystem. <p>Host BASH scripts are useful for setting up mirrors or updating DNS servers.</p> <p>The BASH scripts listed are executed in the order specified.</p> <p>The error code returned by each BASH script is validated, and NVIDIA Build-FS will terminate upon the first error and report a message for debugging.</p>
Example(s)	<pre> "PreInstalls": { "\${BUILD_FS_DIR}/mirror-setup/setup-mirror.sh": "host", "setup-dns.sh": "target", "\${HOME}/crypt_checker.sh": "target_copy" } </pre>
Field	CopyTargets

Field	Base
Type	Array
Acceptable Values	<p><i>Items type:</i> String</p> <p><i>Items content:</i> Valid UNIX filepaths to CopyTarget BASH scripts or MANIFESTS.</p> <p>or</p> <p><i>Items type:</i> dict</p> <p><i>Items content:</i></p> <pre>{ "Manifest": "<path to copytarget yaml/script> (String) (Required)", "NvWorkspace": "<path from which files listed in copytarget are copied> (String) (Optional)", "SourceType": "<Copytarget source type> (String) (Optional)", "Args": { "Add": "<args to be added from copytarget cmdline> (String) (Optional)", "Del": "<args to be deleted from copytarget cmdline> (Str) (Optional)" } (Optional) }</pre>
Req	No
Default	None
Instructions	<p>Specifies a list of zero, one, or more CopyTarget to be executed on the host during Build stage.</p> <p>See CopyTarget Documentation for more details.</p> <p>The list of CopyTarget are executed in the order specified.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> >Note: <p>If item type is dict, to avoid overriding certain fields, do not specify the field in the dict.</p> </div>

Field	Base
	<pre> "CopyTargets": ["\${WORK_DIR}/copytargets/copytarget-libraries.yaml", "\${WORK_DIR}/copytargets/copytarget-tools.yaml", { "Manifest": "\${WORK_DIR}/copytargets/copytarget-sdk.yaml", "NvWorkspace": "/usr/local/bin/sdk/", "SourceType": "custom" }, { "Manifest": "\${WORK_DIR}/copytargets/copytarget-sdk.yaml", "SourceType": "custom", "Args": { "Add": "--string-option1 val1 --bool-option2", "Del": "--string-option3 --bool-option2" } }] </pre>
Example(s)]
Field	PostInstalls
Type	Object
Acceptable Values	<p><i>Key content:</i> Valid UNIX filepath to a BASH script.</p> <p><i>Value type:</i> String</p> <p><i>Value content:</i> enum[host, target, target_copy]</p>
Req	No

Field	Base
Default	None
Instructions	<p>Specifies a list of zero, one, or more BASH scripts to be executed on the host or virtualized target during the Post-Build stage.</p> <p>For each object element, the value of the element specifies whether the specific script is to be executed on the host or virtualized target. A value of:</p> <ul style="list-style-type: none"> "host" is the indicator to execute existing BASH script on the host. "target" is the indicator to execute existing BASH script on the virtualized target. "target_copy" is the indicator to copy existing BASH script on the host to /tmp/ of the target filesystem. execute the script in /tmp/ on the virtualized target. remove the script in /tmp/ of the target filesystem. <p>Host BASH scripts are useful for setting up mirrors or updating DNS servers.</p> <p>The BASH scripts listed are executed in the order specified.</p> <p>The error code returned by each BASH script is validated, and NVIDIA Build-FS will terminate upon the first error and report a message for debugging.</p>
Example(s)	<pre> "PostInstalls": { "\${BUILD_FS_DIR}/mirror-setup/teardown-mirror.sh": "host", "teardown-dns.sh": "target", "\${HOME}/crypt_priv_files.sh": "target_copy" } </pre>
Field	FilesystemCleanup
Type	Array

Field	Base
Acceptable Values	<p><i>Items type:</i> String</p> <p><i>Items content:</i> Valid UNIX filepaths</p>
Req	No
Default	None
Instructions	Specifies a list of zero, one, or more target files to be deleted as part of Post-Build stage.
Example(s)	<pre>" FilesystemCleanup ": ["/tmp/temp.txt", "/var/tmp/tmp.log"]</pre>
Field	ImageSize
Type	String
Acceptable Values	64-bit Unsigned Integer enclosed in double quotes.
Req	No
Default	8589934592
Instructions	<p>The size of the partition to which the current image will be expanded to (defaults to 8GB).</p> <p>This is required for determining values Filesystem Metadata values like journal size, inode_size, inode_byte_ratio.</p> <p>This will help create the required number of inodes and journal in the smaller filesystem image generated,</p> <p>considering the image file expansion.</p>

Field	Base
Example(s)	"ImageSize": "4294967296"
Field	FilesystemType
Type	String
Acceptable Values	String must be using characters from the set [A-Za-z0-9_-]
Req	No
Default	standard
	Specifies the type of filesystem to be created for the CONFIG. This option is forwarded to copytarget while executing the copytarget manifests, and instructs copytarget whether to copy the file, based on rules written inside the copytarget manifest.
Instructions	See CopyTarget Documentation for more details.
Example(s)	"FilesystemType": "standard"
Field	Mirrors
Type	Array

Field	Base
Acceptable Values	<p><i>Items type:</i> String</p> <p><i>Items content:</i> Valid Debian mirrors</p> <p>or</p> <p><i>Items type:</i> dict</p> <p><i>Items content:</i></p> <pre>{ "Path": "<Valid Unix filepath> (String)", (Required) "Type": "enum[\"local_debian_mirror\", local_debian_folder", "debian"] (String)" (Required) }</pre> <p>or</p> <pre>{ "Path": "<Valid Debian mirror> (String)", (Required) "Type": "enum[\"debian_mirror\"] (String)" (Required) }</pre>
Req	No
Default	None

Field	Base
Instructions	<p>Specifies zero, one, or more Debian mirrors hosting Debians specified in <code>DebianPackages</code>.</p> <p>If the value is a 'string', ensure these mirror values adhere to the package manager's syntax of mirrors.</p> <p>If the value is a dict, then depending on the value of 'Type', the file path in 'Path' is treated differently:</p> <p><code>local_debian_folder</code>:</p> <p>Build-FS shall copy the ".deb" files from 'Path' directory to a temporary mirror path in the target filesystem "/var/mirror-<n>"</p> <p>Build-FS shall create a local Debian mirror with the files in /var/mirror-<n></p> <p>Build-FS shall edit /etc/apt/sources.list of the target to consider /var/mirror-<n> as a local Debian mirror while installing packages.</p> <p>Build-FS shall remove these Debians while cleaning up before generating the filesystem image.</p> <p><code>local_debian_mirror</code>:</p> <p>Build-FS shall treat 'Path' as a valid Debian mirror</p> <p>Build-FS shall mount this path to the target filesystem at a mount point /mnt/<n></p> <p>Build-FS shall edit /etc/apt/sources.list of the target to consider /mnt/<n> as a local Debian mirror while installing packages.</p> <p>Build-FS shall umount these paths while cleaning up before generating the filesystem image.</p> <p><code>debian</code>:</p> <p>Build-FS shall treat the file pointed by 'Path' as a valid Debian Mirror installer.</p> <p>Build-FS shall install the Mirror installer in the target filesystem.</p> <p>Build-FS relies on the installer Debian to contain valid mirror configuration files in /etc/apt/sources.list.d/. These file contents are appended to /etc/apt/sources.list by Build-FS before installing Debian packages.</p> <p>Build-FS shall purge the Mirror installer Debian before generating the filesystem image.</p> <p><code>debian_mirror</code>:</p> <p>Contents of 'Path' is appended to /etc/apt/sources.list.</p> <p>Ensure these mirror values adhere to the package manager's syntax of mirrors.</p> <p>Ordering of Debian mirrors are preserved.</p> <p>If the field is not provided, target filesystem will not be configured with any apt mirrors.</p>

Field	Base
	<pre>"Mirrors": ["deb http://ports.ubuntu.com/ubuntu-ports/ bionic main universe restricted", { "Type": "debian", "Path" : "/tmp/mirror-installer.deb"}, { "Type": "local_debian_mirror", "Path": "/opt/deb_mirror/"}, { "Type": "local_debian_folder", "Path" : "/opt/deb_folder/"}, { "Type": "debian_mirror", "Path": "deb http://ports.ubuntu.com/ubuntu-ports/ bionic-updates main universe restricted"}</pre>
Example(s)]
Field	Users
Type	Object

Field	Base
	<p><i>Key content:</i> Unique user identification alphanumeric string</p> <p><i>Value type:</i> dict</p> <p><i>Value content:</i></p> <pre data-bbox="425 508 1192 967">{<key>: { "UID" : "<User-Identifier in target /etc/passwd>" (String) (Required), "Password": { "HashedPassword" : "<Hashed-Password>" (String) (Required) }, "Username" : "<Username in target /etc/passwd>", (String) (Required) }}</pre> <p>Acceptable Values Where UID must be a valid UNIX user identifier and password must be a valid UNIX password.</p>
Req	No
Default	None

Field	Base
Instructions	<p>Specifies a list of zero, one, or more users to be added to the target filesystem.</p> <p>For each key-value pair in the JSON dictionary, a user account is described by its attributes: user-identifier, username, password entry. This entry will either add and update the user account as described by workflow below. The ordering of users to be added is the order of dictionary entries.</p> <p>Ensure usernames, user identifiers, and user passwords are valid UNIX standards.</p> <p>User-identifier is recommended to be above 1000, and not conflicting with existing user identifiers; otherwise, conflicting user-identifiers will be reported as an error.</p> <p>The password entry (i.e. value for the key "Password") is a dictionary with key "HashedPassword" and value hashed-password created using unix crypt algorithm. The alternate way to provide password using plain text is also supported but not preferred due to security reasons as shown below:</p> <pre>"Password": "<Password text>"</pre> <p>Workflow:</p> <p>Build-FS fetches the user entry's UID attribute in the CONFIG file and then checks if there exists a user account whose identifier matches value of UID in the filesystem described by option "-f" or "Base".</p> <p>If the user account exists, Build-FS updates the username of that account to what is specified in the user entry of the CONFIG file.</p> <p>If the user account does not exist, Build-FS proceeds to add the user account with the username provided in the user entry of the CONFIG file.</p> <p>In all the cases, the Build-FS sets the user account password to the value corresponding to the password attribute in the user entry of the CONFIG file.</p> <p>Note that Build-FS updates the username's user-group when updating the username of the user account. For example: when renaming username nvidia, with user-group nvidia and UID 1000 to username nvidia2, Build-FS updates the user's group nvidia to nvidia2 keeping same UID 1000.</p> <p>Note that the previous version of Users block where one entry is : "<username>" : ["user-id", "password"], continues to be supported to only add users for backward compatibility but not to update users.</p>

Field	Base
	<pre>"Users": { "one": { "UID": "1001", "Username" : "nvidia2", "Password": "driveos" }, "two": { "UID": "1002", "Username" : "nvidia3", "Password": { "HashedPassword": "\$6\$4bhqDdYb\$kxJApfqarvpuMhLweydYp7 .NqSFXWxML8N3JywqadmlEp9GF89553PNBAYBTdGmfBaUe7/7LxpP8PBBQICJT70" }, }, }</pre>
Example(s)	
Field	Groups
Type	Object

Field	Base
	<p>Key content: Unique user identification alphanumeric string</p> <p>Value type: dict</p> <p>Value content:</p> <pre>{ "<key >": { "GID" : "<Group-Identifier in target /etc/group>" (String) (Required), "Groupname" : "<Group name in target /etc/group>", (String) (Required) } }</pre> <p>Acceptable Values Where GID must be a valid UNIX group identifier and groupname must be a valid group name string.</p>
Req	No
Default	None
Instructions	<p>Specifies a list of zero, one, or more groups to be added to the target filesystem.</p> <p>For each key-value pair in the JSON, a group is described by its attributes: groupname and the group-identifier. This entry will either add or update the group as described by workflow below. Ordering of groups to be added is preserved.</p> <p>Ensure group names and group identifiers are valid UNIX standards.</p> <p>Group identifier is recommended to be above 1000, and not conflicting with existing group identifiers; otherwise, conflicting user identifiers will be reported as an error.</p> <p>Workflow:</p> <p>Build-FS fetches the group entry's GID attribute in the CONFIG file and then checks if there exists a group whose identifier matches value of GID in the filesystem described by -f option or "Base".</p> <p>If the group exists, Build-FS updates the groupname of that account to what is specified in the group entry of the CONFIG file.</p> <p>If the group does not exist, Build-FS proceeds to add the group with the groupname provided in the group entry of the CONFIG file.</p> <p>Note that the previous version of Groups block where one entry is: "<groupname>": "<group-id>", continues to be supported to only add groups for backward compatibility but not to update groups.</p>

Field	Base
	<pre>"Groups": { "one": { "Groupname": "automotive", "GID": "2001" } }</pre>
Example(s)	}
Field	Memberships
Type	Object
Acceptable Values	<p><i>Key content:</i> valid UNIX username</p> <p><i>Value type:</i> Array</p> <p><i>Value content:</i> [group1(String),..., groupN(String)]</p> <p>Where group* must be a valid UNIX groupname or group identifier.</p>
Req	No
Default	None
Instructions	Specifies a list of zero, one, or more group memberships to be added to the target filesystem.

Field	Base
	<pre>"Memberships": { "nvidia": ["1000", "audio", "cdrom", "dialout",] }</pre>
<hr/>	
Field	DebianPackages
Type	Array
Acceptable Values	<p><i>Items type:</i> String</p> <p><i>Items content:</i> Valid Debian package names.</p>
Req	No
Default	None
Instructions	<p>Specifies a list of zero, one, or more Debian packages to be installed on target filesystem during Build stage.</p> <p>Each Debian package can be accompanied by a valid corresponding package version but is entirely optional. If no package version is specified, the latest version as reported by the Debian mirror will be acquired and installed</p> <p>If the list of Debians and or the specified versions conflict, an error will be reported.</p>

Field	Base
Example(s)	<pre>" DebianPackages ": ["openssh-server=1-4ubuntu0.3", "vim"]</pre>
Field	Hostname
Type	String
Acceptable Values	Valid UNIX hostname
Req	No
Default	None
Instructions	<p>Host name of the target filesystem is updated to the one specified as the value for HostName.</p> <p>If HostName Field is not present in the CONFIG, the host name of the Base filesystem is preserved.</p>
Example(s)	"HostName": "auto-ubuntu"
Field	Mount

Field	Base
Acceptable Values	<pre>Value type: array{ "<Path of mount point> (Required) (String)": { "Type": "<Type of the Linux Filesystem to be mounted> (Required) (String)", "MountOptions": "<Mount specific options> (Required) (String)", "Device": "<Path to the device node or the partition to mount> (Required) (String)" }, }</pre> <p>The values to the Mounts Field is an array of dictionaries. Each dictionary entry's key is the path to the Mount point which the directory the mount the storage device or the partition.</p> <p>The dictionary entry's value following the key is the dictionary of 3 key-value pairs where keys are {"Type", "MountOptions", "Device"}.</p> <p>The Type is the filesystem type such as ext4, ext3, ext2, nfs, overlayfs, ubifs and so on.</p> <p>MountOptions are the comma-separated specific mount options as required. If no specific options are required, it must be assigned as "defaults".</p> <p>Device is the path to the device node of the partition that contains the filesystem in Type and must be mounted on the mount-point (referred above) with the options from MountOptions.</p>
Req	No
Default	None
Instructions	The Mounts entry recorded in Build-FS config is used to generate an /etc/fstab entry and append it to be filesystem's /etc/fstab in the same order as listed in the Mounts block. The block must be populated only when the user is sure of the partition/device in Device contains the filesystem of Type. Entering a Mounts entry without an existing filesystem in the Device leads to a mount error during the filesystem runtime.
Example(s)	<pre>"Mounts": { "/home/": { "Type": "ext4", "MountOptions": "defaults", "Device": "/dev/vblkdev3" }, }</pre> <p>This example mounts the ext4 filesystem in /dev/vblkdev3 at /home/ with default options.</p>

Field	Base
Field	FilesystemInclude
Acceptable Values	<p>Value type: array{ ..., "<Path to file to be included> (Required) (String)", ..., "<Path to directory to be included>/ (Required) (String)", ...}</p> <p>The FilesystemInclude block contains the subset of files and directory to be chosen from</p> <ul style="list-style-type: none"> the Build-FS filesystem working directory after building the filesystem content. The block is optional and if it is absent, the entire Build-FS filesystem working directory goes into the filesystem image. The path to files and the path to directories slightly differ because the path to directory always must end with a slashes to indicate a directory where as the path of a file must not end with a slash. <p>The list of paths required must be expressed in absolute path directory in the Build-FS config or defined using intermediate environment variables and in such cases the environment must define variable.</p>
Default	None
Instructions	The FilesystemInclude block applies when user wishes to copy only a subset of data from filesystem work directory to the final image. However, if the user wants the entire content in the workspace, FilesystemInclude block must to be used.
Example(s)	<pre>"FilesystemInclude": ["/etc/group", "/etc/gshadow", "/etc/passwd", "/etc/shadow", "/etc/subgid", "/etc/subuid", "/etc/passwd", "/home/"],</pre> <p>This example ensures the files listed (until and excluding /home) are chosen along with /home/ directory and its contents are the subset of data that shall be part of the Build-FS created filesystem image.</p>
Field	AssociatedFilesystems

Field	Base
Acceptable Values	<pre>Value type: array{ ... "<Path to build-fs CONFIG associated with the parent CONFIG OR base filename of build-fs CONFIG> (Required) (String)", "<Path to build-fs CONFIG associated with the parent CONFIG OR base filename of build-fs CONFIG> (Required) (String)", ... }</pre>
Req	No
Default	None
Instructions	<p>The Build-FS CONFIG in AssociatedFilesystems block can either be an entry to the absolute path of the CONFIG file or just a filename. If it is a filename, then Build-FS looks for the CONFIG in Associated Filesystems in the parent CONFIG's directory.</p> <p>The Build-FS CONFIG in AssociatedFilesystems block is a valid Build-FS config (no different than the associator's CONFIG). In fact, each CONFIG in the AssociatedFilesystems can be executed as dedicated instances of Build-FS. The AssociatedFilesystems block can be applied recursively such that the Build-FS CONFIG within the AssociatedFilesystems can request more AssociatedFilesystems. Finally, the fields within CONFIG of AssociatedFilesystems are not flattened into the parent Build-FS CONFIG's MANIFEST but instead each CONFIG in AssociatedFilesystems creates its corresponding MANIFEST.</p>
Example(s)	<pre>"AssociatedFilesystems": ["driveos-core-rfs-user-metadata.CONFIG.json", "\${BUILD_FS_DIR}/configs/driveos-core-rfs-user-data.CONFIG.json"], This example the first associated filesystem shall be located at the same directory as the parent's CONFIG whereas the final CONFIG must be present at the full path "\${BUILD_FS_DIR}/configs/driveos-core-rfs-user- data.CONFIG.json".</pre>
Field	SELinux

Field	Base
Acceptable Values	<p>Value type: dictValue content:{</p> <p>"SetFiles": <Path to the 3rdparty SELinux setfiles application> (String) (Required),</p> <p>"PolicyFile": <Path to the SELinux policy binary file> (String) (Required),</p> <p>"ContextFile": <Path to SELinux context file> (String) (Required)}</p> <p>The value is of type dictionary but the set of keys in the dictionary is fixed to {"SetFiles", "PolicyFile", "ContextFile"}.</p> <p>The SetFiles key holds the absolute path to host-side setfiles application. This application is provided Ubuntu host-side package check policycoreutils.</p> <p>The PolicyFile key holds the absolute path to the SELinux precompiled policy binary file (built using checkpolicy application).</p> <p>The ContextFile key holds the absolute path to SELinux context file. For more information on SELinux policy and SELinux context file, please refer to DRIVE OS LINUX SELinux documentation.</p>
Req	No

Field	Base
Default	None
Instructions	<p>The SELinux block specifies the required inputs for build-fs to apply appropriate the SELinux attributes (SELinux attributes is a part of extended attributes of a file or a directory). SELinux block is a dictionary but Build-FS only supports and requires 3-fixed keys {"SetFiles", "PolicyFile", "ContextFile"}. The value of each of the keys must be the respective absolute paths. Build-FS supports specifying the paths using environment variables only if such variables are defined the Build-FS environment file.</p> <p>For DRIVE OS LINUX, the setfiles application can be obtained by installing policycoreutils package via apt-get. Then, the app resides at path /sbin/setfiles. The workflow of editing policy and context file is documented in DRIVE OS LINUX SELINUX documentation.</p>
Example(s)	<pre> 1) Explicit path to each of the entries."SELinux": { "SetFiles": "/sbin/ setfiles", "PolicyFile": "\${NV_WORKSPACE}/drive-linux/filesystem/contents/configs/ selinux/policy.30", "ContextFile": \${NV_WORKSPACE}/drive-linux/filesystem/contents/configs/ selinux/file_contexts" } 2) Paths can be defined using build-FS environment variable"SELinux": { "SetFiles": "\${SELINUX_SETFILES_PATH}", "PolicyFile": "\${SELINUX_POLICY_PATH}", "ContextFile": "\${SELINUX_CONTEXT_PATH}" }The build-FS environment would contain the expansion of the variables:{... "SDKPLATDIR": "\${NV_WORKSPACE}/drive-linux", "SELINUX_SETFILES_PATH": "/ sbin/setfiles", "SELINUX_POLICY_PATH": "\${SDKPLATDIR}/filesystem/contents/configs/selinux/policy.30", "SELINUX_CONTEXT_PATH": "\${SDKPLATDIR}/filesystem/contents/configs/selinux/ file_contexts", ... }</pre>

7.7.11 Errors

Error	Variable Values
No value for NV_WORKSPACE provided with the -w option.	N/A

Error	Variable Values
NV_WORKSPACE: '<path>' doesn't exist.	path: Value provided after -w in Build-FS command line.
No CONFIG provided with the '-i' option.	N/A
CONFIG file: '<json_file>' doesn't exist.	json_file : Value provided after -i in Build-FS command line.
<config_os> is not supported.	config_os : Value of "OS" tag in the Build-FS CONFIG file.
<variable> is not defined.	variable: Any required Build-FS environment variable.
Command returned non-zero error code: <cmd_string>	cmd_string : Command line used to execute binary outside Build-FS python scripts.
Unsupported execution location: '<run_target>'. Expecting a value from the list: [host, target, target_copy]	run_target : Runtime target configured for Pre-Install/ Post-Install scripts.
Unknown Base file format: '<base>', Please provide Base in a Build-FS supported format.	base: Value of "Base" tag in the Build-FS CONFIG file.

Error	Variable Values
Required Field: '<field>' absent from input CONFIG file: '<config>'.	<p>field:</p> <p>Field is one of the required fields of Build-FS CONFIG.</p> <p>(See NVIDIA Build-FS CONFIG Semantics for more information).</p> <p>config:</p> <p>Input CONFIG file provided to Build-FS.</p>
<field>: Value is not a string in the input CONFIG file: '<config>'.	<p>field:</p> <p>Field is one of the required fields of Build-FS CONFIG.</p> <p>(See NVIDIA Build-FS CONFIG Semantics for more information).</p> <p>config:</p> <p>Input CONFIG file provided to Build-FS.</p>
<field>: Value is not a list in the input CONFIG file: '<config>'.	<p>field:</p> <p>Field is one of the required fields of Build-FS CONFIG.</p> <p>(See NVIDIA Build-FS CONFIG Semantics for more information).</p> <p>config:</p> <p>Input CONFIG file provided to Build-FS.</p>

Error	Variable Values
<field>: Value is not a dict in the input CONFIG file: '<config>' 	field: Field is one of the required fields of Build-FS CONFIG. (See NVIDIA Build-FS CONFIG Semantics for more information). config: Input CONFIG file provided to Build-FS.
FileNotFoundException: [Errno 2] No such file or directory: '<file>' 	file: Missing file in the host system where Build-FS is running.
Usage: build_kit.py [options] build_kit.py: error: no such option: <incorrect_option> 	incorrect_option: Incorrect option provided in Build-FS command line
Usage: build_kit.py [options] build_kit.py: error: option --create-tar: invalid choice: '<val>' (choose from 'yes', 'no') 	val: Value provided to --create-tar option.
Usage: build_kit.py [options] build_kit.py: error: option --create-image: invalid choice: '<val>' (choose from 'yes', 'no') 	val: Value provided to --create-image option.

7.7.12 Examples

7.7.12.1 To add users, groups, and memberships to the filesystem

This example demonstrates how to:

1. Add user "nvidia2" with identifier "1001" and password "driveos".
2. Add group "automotive" with identifier "2001".
3. Assign membership between user "nvidia2" and group "automotive".

```
{
  "Users": {
    "one": {
```

```

        "UID": "1001",
        "Username" : "nvidia2",
        "Password": "driveos"
    }
},
"Groups": {
    "one": {
        "Groupname": "automotive",
        "GID": "2001"
    }
},
"Memberships":
{
    "nvidia2":
    [
        "automotive"
    ]
}
}
}

```

7.7.12.2 To update existing user, group and set passwd in the filesystem

This example demonstrates how to:

1. We assume the filesystem has a user and group from above example:
 - a. A user with username "nvidia2", user-id "1001" and password "driveos".
 - b. A group "automotive" with group-id "2001".
2. Update user to username "nvidia3" and keep identifier "1001" and set password to "nvidia3".
3. Update group to groupname "auto" and keep group identifier "2001".
4. The memberships (which are based on UID/GID) are automatically updated in the filesystem.

```

{
    "Users": {
        "one": {
            "UID": "1001",
            "Username" : "nvidia3",
            "Password": "nvidia3"
        }
    },
    "Groups": {
        "one": {
            "Groupname": "auto",
            "GID": "2001"
        }
    }
}

```

7.7.12.3 To set password to given value securely using hashed-password to the filesystem

This example demonstrates how to:

1. We assume the filesystem has a user and group from above example:
 - a. A user with username "nvidia2", user-id "1001".
2. Set password "nvidia2" for the username "nvidia2".
3. When string "nvidia2" is hashed using crypt with salt "4bhqDdYb" it produces password in the entry below. Tool mkpasswd can be used to get hashed-password.

```
{
  "Users": {
    "one": {
      "UID": "1001",
      "Username" : "nvidia2",
      "Password": {
        "HashedPassword": "$6$4bhqDdYb$KxJApfqarvpuMhLweydYp7.
NqSFXWxML8N3Jywqadm1Ep9GF89553PNBAYBTdGmfBaUe7/7LxpP8PBBQ1CJT70"
      },
    }
  }
}
```

7.7.12.4 To install Debian packages from Ubuntu mirrors

This example demonstrates how to install Debian packages "vim" and "nano" from Ubuntu mirror "deb http://ports.ubuntu.com/ubuntu-ports/ bionic main universe restricted".

```
{
  "Mirrors": [
    [
      "deb http://ports.ubuntu.com/ubuntu-ports/ bionic main universe restricted"
    ],
    "DebianPackages": [
      [
        "vim",
        "nano"
      ]
    ]
}
```

7.7.12.5 To copy files to the target filesystem

This example demonstrates how to:

```
{
  "CopyTargets": [
    [
      "${WORK_DIR}/copytarget-nvidia.yaml"
    ]
}
```

The CopyTarget file `copytarget-nvidia.yaml` is used in the example below:

```
version: '1.4'
fileList:
  - destination: /home/nvidia/nvidia.txt
    source: nvidia.txt
    perm: 644
    owner: root
    group: root
```

7.7.12.6 To run preinstall and postinstall scripts

This example demonstrates how to:

```
{
  "PreInstalls":
  {
    "${WORK_DIR}/preinstall.sh": "target_copy"
  },
  "PostInstalls": {
    "${WORK_DIR}/postinstall.sh": "target_copy"
  }
}
```

The `preinstall.sh` script is used in the example below:

```
#!/bin/sh
echo "nameserver.nvidia.com" >> /etc/resolv.conf
```

The `postinstall.sh` script is used in the example below:

```
#!/bin/sh
sed -i "s/nameserver.nvidia.com//g" /etc/resolv.conf
```

7.8 CAN Driver

Linux releases support two instances of NVIDIA Tegra Controller Area Network (TegraCAN). The CAN controller for Tegra family devices is the Bosch MTTCAN controller IP.

TegraCAN provides two SocketCAN interfaces in the Linux kernel. The SocketCAN interface is similar to network interfaces in Linux kernel. SocketCAN documentation is available in Linux kernel documentation online at:

```
<Top>/drive-linux/kernel/source/oss_src/kernel/Documentation/devicetree/bindings/net/
can/
```

For information about SocketCAN limitations, see the Release Notes.

There are two implementations of MTTCAN SocketCAN driver:

- `mttcan` driver that controls CAN controller directly through CPU running Linux

- > `mttcan-ivc` that uses CAN driver services through IVC and implemented in SPE-FW. This driver can potentially be used by both SPE firmware and Linux.

The selection of either `mttcan` or `mttcan-ivc` is done through device tree solution. Ensure that both drivers for a given CAN controller are not simultaneously enabled through device tree.

7.8.1 Enabling CAN Driver in Linux Kernel

You must enable the CAN driver using the following procedures.

7.8.1.1 To enable SPE based mttcan_ivc driver

1. For the targeted board device tree include file, make the following device tree changes. The supported boards include p3710-10-a01 and p3663-a01 . For custom board configurations, make the following changes.

- > Enable `mttcan0-ivc` and `mttcan1-ivc` in the target device tree:

```
mttcan0-ivc {
    status = "okay";
};
mttcan1-ivc {
    status = "okay";
};
```

- > Disable `mttcan` driver in the target device tree:

```
mttcan@c310000 {
    ...
    status = "disabled";
;
mttcan@c320000 {
    ...
    status = "disabled";
};
```

2. Make sure that SPE-FW is loaded by enabling the `-L` flash option. Ensure that your `global_storage.cfg` has SPE and Warmboot binaries enabled.

7.8.1.2 MTTCAN as a Module

Before running MTTCAN as a module, the user must ensure it is loaded on the target.

7.8.1.2.1 To install modules

1. On the target, determine whether following modules are loaded, by running the `lsmod` command in the Linux shell.

```
can.ko
can-dev.ko
mttcan.ko
mttcan_ivc.ko
```

2. If the modules are not loaded, determine whether they are in following, directory:

```
/lib/modules
```

3. If the modules are not in this directory, set the following CONFIG options in defconfig and rebuild kernel.

```
CONFIG_CAN=m
CONFIG_CAN_RAW=m
CONFIG_CAN_DEV=m
CONFIG_MTTCAN=m
CONFIG_MTTCAN_IVC=m
```

4. Alternatively, these options can be set using the *instructions* described in Compiling the Kernel in the *NVIDIA DRIVE OS 6.0 Linux PDK Development Guide*, "To compile the kernel" topic, steps 1-3 to:

- > Setup the environment macros
- > Set the kernel source directory
- > Create an output directory and configure the kernel for the board

5. Execute the following command.

```
make -C kernel O=${PWD}/out-linux
DEFCONFIG_PATH=$PWD/t23x/arch/arm64/configs menuconfig
[*] Networking support --->
    <M> CAN bus subsystem support --->
    <M> Raw CAN Protocol (raw access with CAN-ID filtering)
    <M> CAN Device Drivers --->
    <M> Platform CAN drivers with Netlink support
    <M> Bosch M_TTCAN Devices
    <M> Bosch M_TTCAN IVC Devices
```

6. Exit and save.

7. Rebuild the kernel and copy the kernel Image from the kernel out directory to:

```
drive-linux/kernel
```

For more information, see Compiling the Kernel in the *NVIDIA DRIVE OS 6.0 Linux PDK Development Guide*.

8. Reflash the target.

For more information, see [Flashing](#).

9. Load all required modules.

```
# modprobe mttcan
# modprobe mttcan_ivc
```

This command assumes module dependencies are properly set.

10. If the modprobe command fails, manually load the modules:

```
insmod /lib/modules/<KERNEL_VERSION>/kernel/net/can/can.ko
insmod /lib/modules/<KERNEL_VERSION>/kernel/drivers/net/can/can-dev.ko
insmod /lib/modules/<KERNEL_VERSION>/t18x/drivers/staging/mttcan/mttcan.ko
```

```
insmod /lib/modules/<KERNEL_VERSION>/t18x/drivers/net/can/mttcan /ivc/mttcan_ivc.ko
```

**Note:**

The <KERNEL_VERSION> is in the form of x.x.xx-rtxx-tegra. For example, 4.4.38-rt49-tegra.

7.8.1.3 MTTCAN as a Kernel Built-in Driver

When you build the MTTCAN driver along with the kernel, it is part of the kernel binary.

1. Follow the steps above in [MTTCAN as a Module](#).
2. Set the following CONFIG options in defconfig.

For more information, see *Compiling the Kernel in the NVIDIA DRIVE OS 6.0 Linux PDK Development Guide*.

```
CONFIG_CAN=y
CONFIG_CAN_RAW=y
CONFIG_CAN_DEV=y
CONFIG_MTTCAN=y
CONFIG_MTTCAN_IVC=n
```

**Note:**

Only one of the two configurations can be enabled CONFIG_MTTCAN or CONFIG_MTTCAN_IVC. Both cannot be statically linked simultaneously.

3. Alternatively, these options can be set using the *instructions* described in *Compiling the Kernel in the NVIDIA DRIVE OS 6.0 Linux PDK Development Guide*, "To compile the kernel" topic, steps 1-3 to:
 - > Setup the environment macros
 - > Set the kernel source directory
 - > Create an output directory and configure the kernel for the board
4. Execute the following command.

```
make -C kernel O=${PWD}/out-linux
DEFCONFIG_PATH=${PWD}/t23x/arch/arm64/configs menuconfig
[*] Networking support --->
    <M> CAN bus subsystem support --->
    <M> Raw CAN Protocol (raw access with CAN-ID filtering)
    <M> CAN Device Drivers --->
    <M> Platform CAN drivers with Netlink support
    <M> Bosch M_TTCAN Devices
```

5. Exit and save.
6. Build the kernel and copy the modules from the kernel Image and zImage as follows:

```
cp ${PWD}/out-linux/arch/arm64/boot/zImage
<top>/drive-linux/kernel
cp ${PWD}/out-linux/arch/arm64/boot/Image
<top>/drive-linux/kernel
```

For more information, see Compiling the Kernel in the *NVIDIA DRIVE OS 6.0 Linux PDK Development Guide*.

7. Reflash the target system.

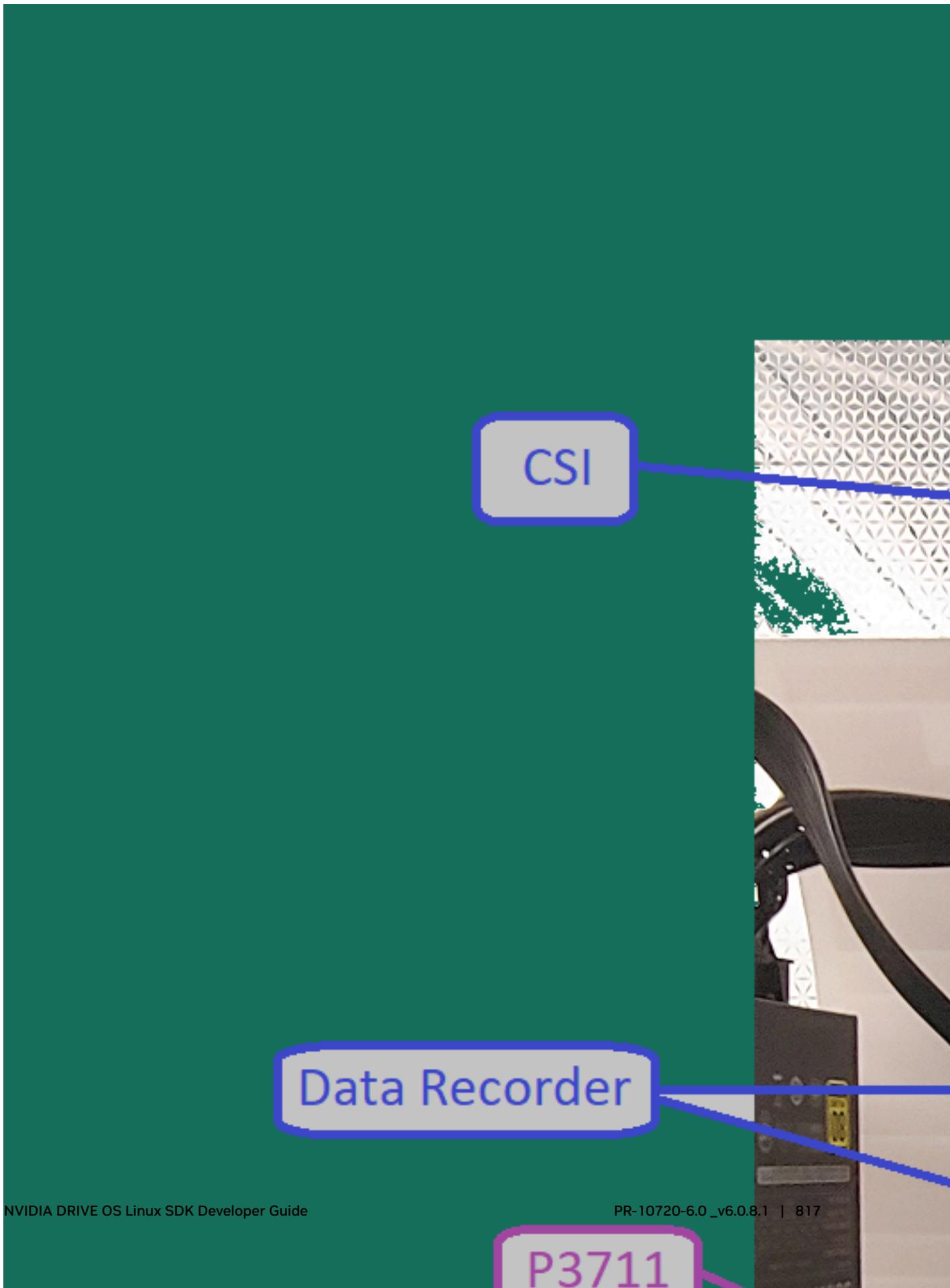
For more information, see [Flashing](#).

7.8.2 Setting Up CAN Loopback

On Orin, you need a special loopback cable with termination to test CAN.

Basic Setup

- > Connect power cable to the board
- > Connect Aurix UART Debug MCU (see Image 1) to host via micro SUB to USB A cable
- > Connect USB-C flashing (see Image 1) to host via USB C to USB A cable



NFS Setup

- > Connect RJ45 EQOS port (see image above) to host via RJ45-RJ45 cable

7.8.2.1 Test Specific Setup



Note: Harness H1/H2 cables are not provided with the devkit and must be purchased separately. Additionally, prepare CAN loopback cable TS3 following the instructions.

The following sections describe how to set up the tests.

AURIX Tests

- > Connect harness H1 to H1/CN2 port (see Image 1)
- > Connect harness H2 to H2/CN1 port (see Image 1)

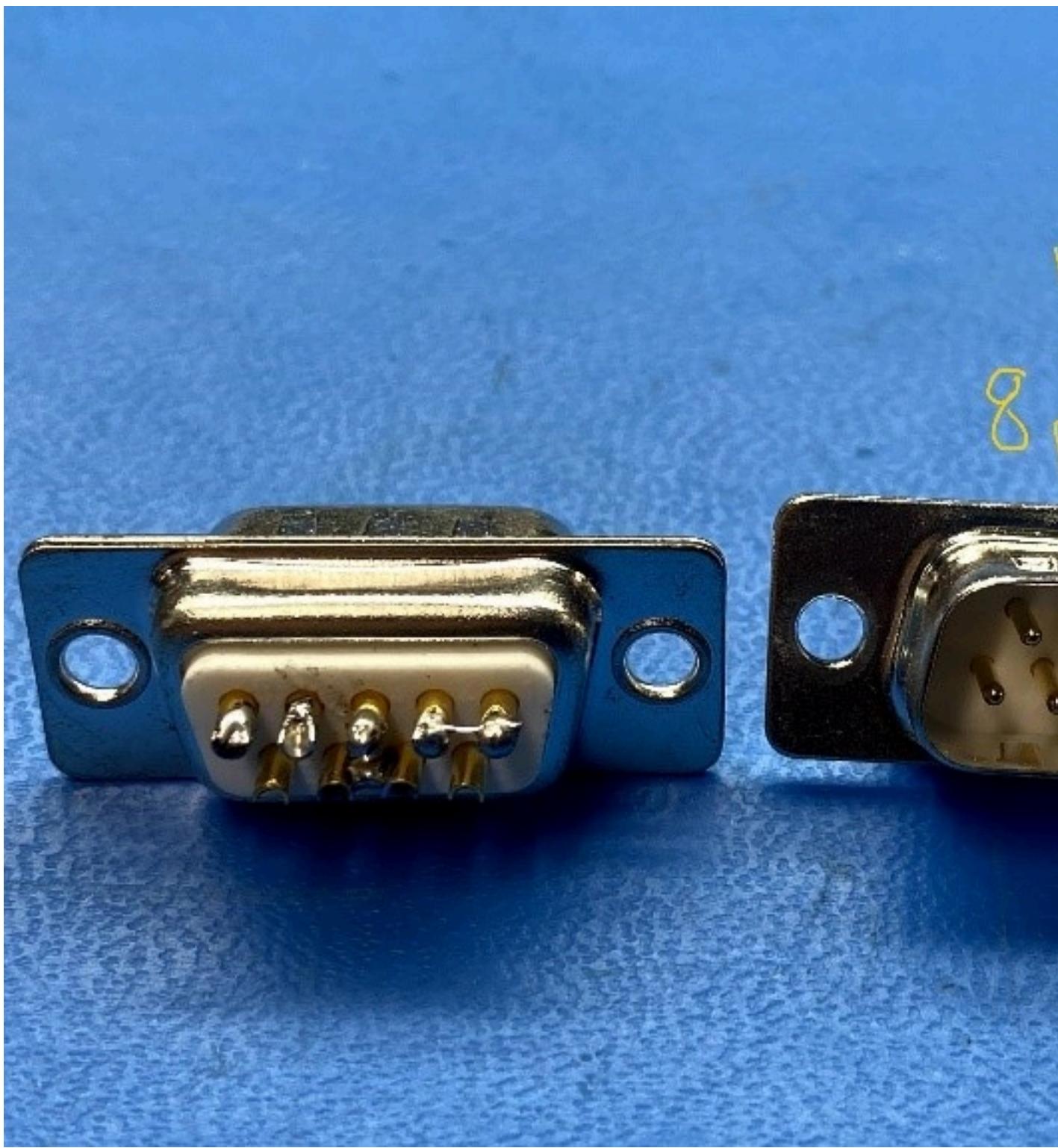
CAN Test

- > Connect CAN loopback connectors to H1/H2 harness as per image below:



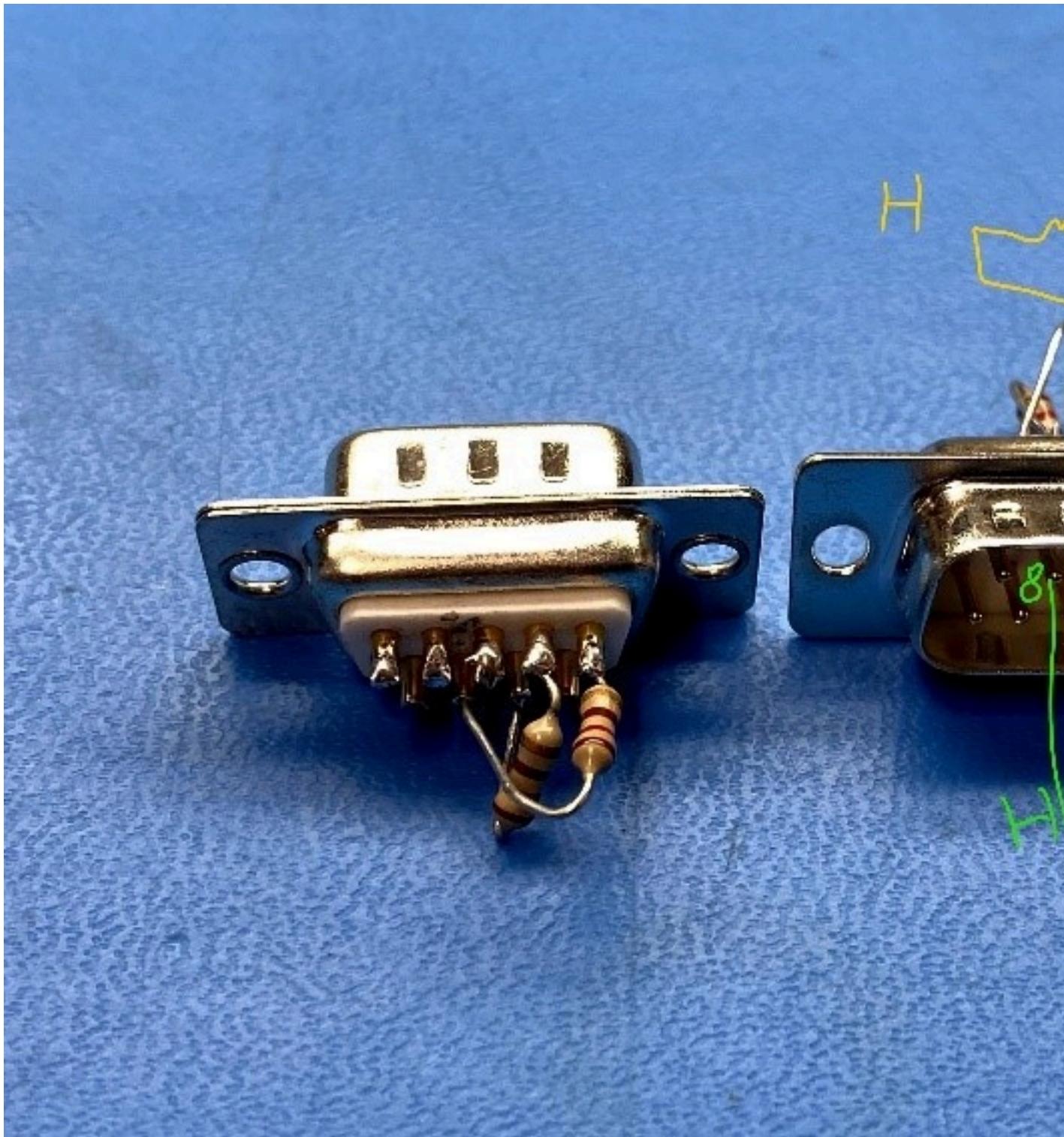
Connector 1: (P3 on harness 2)

Bridge Dongle(Dsub9 Male): Connect pin 1 and 2, connect pin 7 and 8



Connector 2: (P14 on Harness 1)

Termination dongle(Dsub9 Male): connect 120 ohm btw pin 1 and 8 and connect 120 ohm btw pin2 and 7



Connector 3: (P12 - P13 on harness 2 AND P4 of harness 1 to P11 of Harness 2)

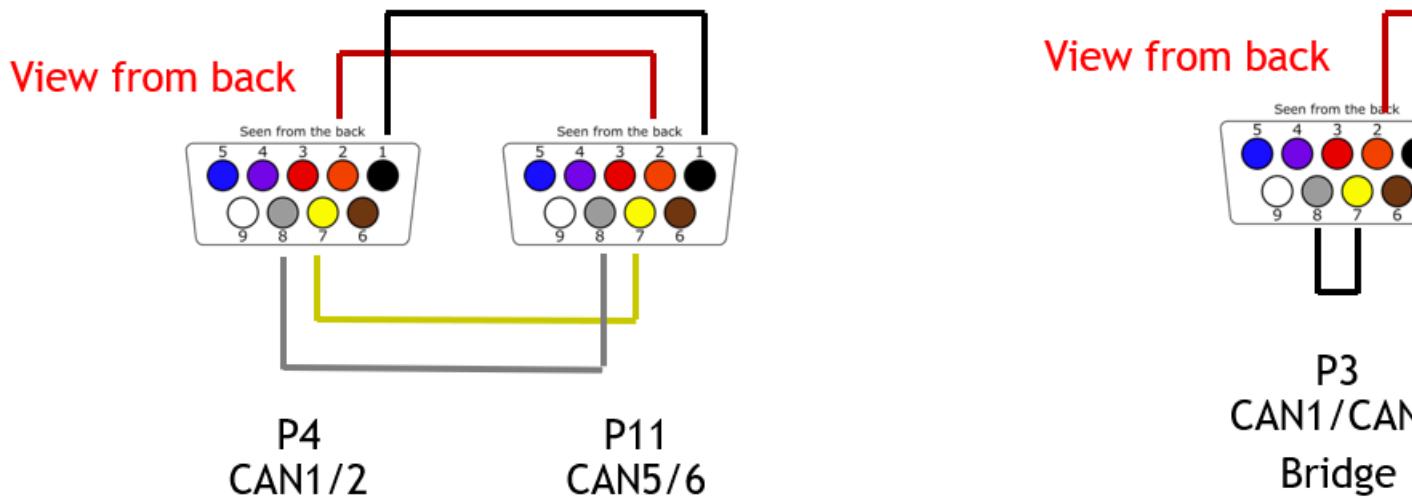
x2 CAN cable(Male to Male): Connect pin 1-pin 1, pin 2-pin 2, pin 7-pin 7, pin 8-pin 8



CAN Loopback Cable TS3

For CAN loopback cable TS3, connect each DSUB 9 connector using the following guidelines and figures:

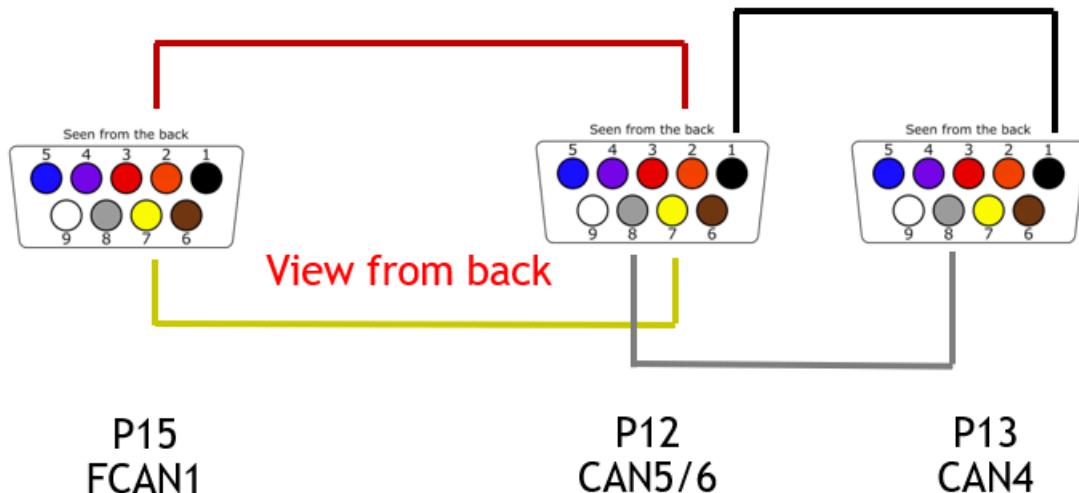
1. Ensure that the wire is 3 inches long between DSUBs.
2. Put the housing on each DSUB.
3. Put a label on each DSUB 9 connector as shown in the following figure.



CAN1<->CAN5 and CAN2<->CAN6 loopback:

- > SMCU_CAN1: P4 - pins 2, 7
- > SMCU_CAN2: P4 - pins 1, 8
- > SMCU_CAN5: P11 - pins 2, 7
- > SMCU_CAN6: P11 - pins 1, 8

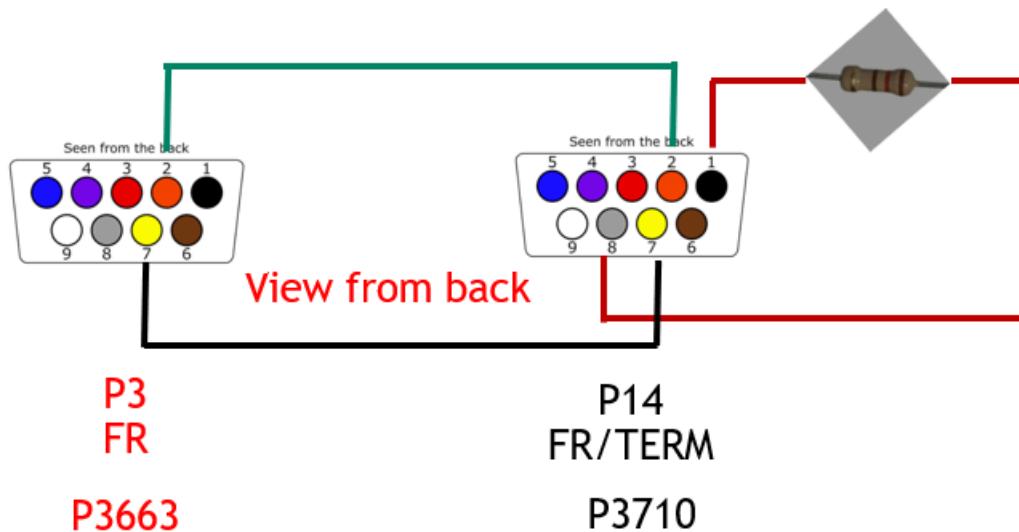
CAN bridge: Short DB9 connector pin1 to pin2 and pin7 to pin8.



CAN5<->FSICAN1 and CAN4<->CAN6 loopback:

- > SMCU_CAN5: P12 - pins 2, 7
- > SMCU_CAN6: P12 - pins 1, 8
- > SMCU_FSICAN1: P15 - pins 2, 7
- > SMCU_CAN4: P13 - pins 1, 8

Can terminator: Connect a 120-Ohm resistor between pin1 and pin8 as shown in the following figure:



Ethernet Tests

Switch Loopback Tests

- > Matenet Ports on 3718 (see image 1)

- Connect port 3 to port 4 via matenet cable to matenet cable
- Connect port 2 to port 5 via matenet cable to matenet cable
- Connect port 1 to port 6 via matenet cable to matenet cable
- > HMTD ports on 3718 (see image 1)
 - Connect J3 to J4 via Quad HMTD to Quad HMTD cable
 - Port J3.1 to Port J4.1
 - Port J3.2 to Port J4.2
 - Port J3.3 to Port J4.3
 - Port J3.4 to Port J4.4

Rework:

- > MGBE 2/3 rework

LAN7431 Test

- > Connect Port HMTD-LAN7431 (see image 1) to HMTD port in E3459
- > Connect RJ45 port of E3459 to host PC

USB Test

- > Connect USB C 3.1 flash drive to port USB-C J8 port (USB-C host port in image 1)
- > Connect USB A 2.0 flash drive to port USB-A J11 port (USB type A (left) in image 1)
- > Connect USB A 2.0 flash drive to port USB-A J23 port (USB type A (right) in image 1)

Rework:

- > USB 2.0 Type A

7.8.3 Setting up SocketCAN Interface

After ensuring `mttcan` module is loaded or the driver is enabled, use following commands to set up the CAN0 network interface. To enable the CAN1 interface, replace `can0` with `can1`.

7.8.3.1 Setting Up the CAN0 Interface

7.8.3.1.1 To set can0 interface bitrate

- > Enter:

```
#ip link set can0 type can bitrate 500000
```

In this command, bitrate can be any valid CAN bitrate for stand CAN.



Note:

125000(125 Kbps), 250000(250 Kbps), 500000(500 Kbps) and 1000000(1Mbps) are supported bitrates for Tegra MTTCAN driver.

Any other bitrate is not validated on Tegra MTTCAN driver.

Only 500000 (500 Kbps) bitrate is supported for Tegra MTTCAN-IVC. Other bitrates are not enabled on Tegra MTTCAN-IVC driver out of box. Details are available in the SPE documentation.

7.8.3.1.2 To get the supported commands

There are various parameters that can be set for CAN interface.

> Enter:

```
# ip link set can0 type can help
```

This command prints the following help:

```
Usage: ip link set DEVICE type can
      [ bitrate BITRATE [ sample-point SAMPLE-POINT] ] |
      [ tq TQ prop-seg PROP SEG phase-seg1 PHASE-SEG1
        phase-seg2 PHASE-SEG2 [ sjw SJW ] ] |
      [ dbitrate BITRATE [ dsample-point SAMPLE-POINT] ] |
      [ dtq TQ dprop-seg PROP SEG dphase-seg1 PHASE-SEG1
        dphase-seg2 PHASE-SEG2 [ dsjw SJW ] ] |
      [ loopback { on | off } ] |
      [ listen-only { on | off } ] |
      [ triple-sampling { on | off } ] |
      [ one-shot { on | off } ] |
      [ berr-reporting { on | off } ] |
      [ fd { on | off } ] |
      [ restart-ms TIME-MS ] |
      [ restart ]
Where: BITRATE  := { 1..1000000 }
       SAMPLE-POINT  := { 0.000..0.999 }
       TQ            := { NUMBER }
       PROP-SEG      := { 1..8 }
       PHASE-SEG1    := { 1..8 }
       PHASE-SEG2    := { 1..8 }
       SJW           := { 1..4 }
       RESTART-MS   := { 0 | NUMBER }
```

7.8.3.1.3 To bring up the can0 interface

> Enter:

```
#ip link set up can0
```

7.8.3.2 Enabling the Flexible Data Rate Mode on MTTCAN

SocketCAN currently supports classic (or standard) CAN mode and Flexible Data (FD) rate. If the devices on your CAN network support FD, the user can enable FD rate mode on MTTCAN.

7.8.3.2.1 To enable FD mode

1. If the interface is already up, disable it.

```
#ip link set down can0
```

2. Set the flexible data rate for the dbitrate option to any valid data rate, provided that dbitrate is greater than the specified bitrate value.

```
#ip link set can0 type can bitrate 500000 dbitrate 2000000 berr-reporting on fd on &&
ip link set up can0
```



Note:

For dbitrate support, iproute package version 2.4.0.0 or later is required. Additionally, the maximum dbitrate support depends on PHY chip on platform. Consult the PHY datasheet to identify and obtain the maximum allowed data bitrate.

dbitrate 1000000 (1 Mbps) and 2000000 (2 Mbps) are supported data bitrates. Any other dbitrate is not validated on TegraMTTCAN driver.

7.8.3.2.2 To check detail statics of the link

- > Enter:

```
#ip -details -statistics link show can0
```

7.8.3.2.3 Miscellaneous Information About OSS SocketCAN Tools

You can use the open source can-utils package to get information about SocketCAN.

```
#apt-get install can-utils
```

7.8.4 How to Test CAN

The following section describes how to test CAN.

7.8.4.1 Test Classic (Non-FD) CAN

1. First, configure both the CAN interfaces with required bitrates:

```
# sudo ip link set can0 type can bitrate 500000 berr-reporting on && sudo ip link set
up can0
# sudo ip link set can1 type can bitrate 500000 berr-reporting on && sudo ip link set
up can1
```

2. Open a new ssh terminal and run the following commands to receive CAN packets sent by can0 interface.

```
# candump -x can0 &
```

3. On another terminal, run the following to send one CAN packet via the can0 interface, where 220 is the CAN ID and 50 is data bytes.

```
# cansend can0 220#50
```

The CAN packet sent by cansend is received on candump terminal.

7.8.4.2 Test FD CAN

1. Configure both the CAN interfaces with required bitrates:

```
# sudo ip link set can0 type can bitrate 500000 dbitrate 2000000 berr-reporting on fd on && ip link set up can0
# sudo ip link set can1 type can bitrate 500000 dbitrate 2000000 berr-reporting on fd on && ip link set up can1
```

2. Open a new ssh terminal and run the following commands to receive CAN packets sent by can0 interface.

```
# candump -x can0 &
```

3. On another terminal, run the following to send one CAN packet via the can0 interface, where 220 is the CAN ID and 50 is data bytes. The 1 after ## denotes that the bit-rate switching (BRS) flag is on.

```
# cansend can0 220##150
```

The CAN packet sent by cansend is received on candump terminal.

7.8.5 CAN Timestamping

The MTTCAN module provides a 16-bit timestamp counter to timestamp received CAN packets in the hardware. At the start of each CAN frame reception, it captures the TSC counter [24-9] bits and stores it alongside the CAN frame. In order to overcome the limitation of 16-bit timestamp, deduce all of the 64 bits of captured timestamp while processing the CAN frame in the interrupt handler. This hardware timestamp is provided to the application using the SocketCAN interface.

Use the candump util to dump the hardware timestamp for each received CAN packet:

```
# candump -ta can0 &
```

7.8.6 Setting up MTTCAN Controller Hardware filters

The CAN Filters interface provided by SocketCAN is software only and is specific to socket. To program hardware filters in the hardware message RAM, the `mttcan` driver exposes following /sys interfaces in Linux kernel.


Note:

Hardware filters are currently supported only for `mttcan` driver and not for `mttcan_ivc` driver.

7.8.6.1 CAN0 sys interfaces

```
/sys/devices/platform/c310000.mttcan/net/can0/std_filter
/sys/devices/platform/c310000.mttcan/net/can0/xtd_filter
/sys/devices/platform/c310000.mttcan/net/can0/gfc_filter
```

7.8.6.2 CAN1 sys interfaces

```
/sys/devices/platform/c320000.mttcan/net/can1/std_filter
/sys/devices/platform/c320000.mttcan/net/can1/xtd_filter
/sys/devices/platform/c320000.mttcan/net/can1/gfc_filter
```

By default, sixteen Standard Message ID Filter and Sixteen Extended Message ID Filter elements are configured in the `mttcan` DT node.

For information on changing the number of filters in the DT node, see Kernel Documentation at:

```
drive-linux/kernel/source/oss_src/kernel/Documentation/i2c
```

7.8.7 Programming Global Filter Configuration

You must configure the global filter using the following procedures.

7.8.7.1 To configure the Global Filter

- > Read/set the values with the following sys interface:

```
/sys/devices/platform/c320000.mttcan/net/can1/gfc_filter
```

7.8.7.2 To get sys interface syntax

- > Enter echo help(h) to the sys interface to obtain the format for the parameters to be provided in dmesg.

```
echo h > /sys/devices/platform/c320000.mttcan/net/can1/gfc_filter
```

The following is an example response from this command.

```
usage:anfs=0..3 anfe=0..3 rrfs=0/1 rrfe=0/1
```

All fields in this register are unsigned integers.

For example:

```
echo "anfs=2 anfe=2 rrfs=0 rrfe=0" > /sys/devices/platform/c320000.mttcan/net/can1/gfc_filter
```

7.8.7.3 To read GFC configuration

- Enter:

```
cat /sys/devices/platform/c320000.mttcan/net/can1/gfc_filter
```

For information on configuring GFC, see section 2.3.20 in the Bosch MTTCAN user manual.

7.8.8 Programming Standard Message ID CAN Filters

If space is reserved for Standard Message ID CAN filters in device tree node, then standard filters can be configured using following sys interface:

```
/sys/devices/platform/c310000.mttcan/net/can0/std_filter
```

- Enter echo help(h) to the sys interface to obtain the format for the parameters to be provided in dmesg.

```
# echo h > /sys/devices/platform/c310000.mttcan/net/can0/std_filter
bash: /sys/devices/platform/c310000.mttcan/net/can0/std_filter: No such file or
directory
```

Where:

- The sfid1 and sfid2 are in HEX.
- The sft, sfec, and idx are in unsinged integers.
- The idx field is optional.
- If index is provided, then the given index filter is updated; otherwise the filters are assigned incrementally.

For example:

```
echo "sft=0 sfec=1 sfid1=123 sfid2=123 idx=0" > /sys/devices/platform/c310000.mttcan/
net/can0/std_filter
```

7.8.8.1 To read standard filter configuration

- Enter:

```
cat /sys/devices/platform/c310000.mttcan/net/can0/std_filter
```

For information on configuring Standard Message ID filters, see section 2.4.5 in the Bosch MTTCAN user manual.

7.8.9 Programming Extended Message ID Filters

If space is reserved in Extended Message ID CAN filters in device tree node, then extended filters can be configured using following sys interface:

```
/sys/devices/platform/c310000.mttcan/net/can0/xtd_filter
```

- > Enter echo help(h) to the sys interface to obtain the format for the parameters to be provided in dmesg.

```
# echo h > /sys/devices/platform/c310000.mttcan/net/can0/xtd_filter
bash: echo: write error: Invalid argument
# [ 3435.907378] net can0: Invalid xtd filter
[ 3435.907384] usage:eft=0..3 efec=0..7 efid1=ID1h efid2=ID2h idx=i
```

Where:

- The efid1 and efid2 are in HEX.
- The eft, efec, and idx are in unsinged integers.
- The idx field is optional.
- If the index is provided, then the given index filter is updated; otherwise the filters are assigned incrementally.

For example:

```
echo "eft=0 efec=2 efid=0x7 efid2=0x21 idx=0" > /sys/devices/platform/c310000.mttcan/
net/can0/xtd_filter
```

7.8.9.1 To read extended filter configuration

- > Enter:

```
cat /sys/devices/platform/c310000.mttcan/net/can0/xtd_filter
```

For information on configuring Extended Message ID filters, see section 2.4.6 in the Bosch MTTCAN user manual.

7.9 Kernel Modules and Limitations

NVIDIA modifies the Linux defconfig to build many drivers as modules, which reduces boot time. The following list shows examples of such modules. Kernel image update requires updated modules from the same build; hence, mixing and matching between kernel/module builds shall result in unpredictable behavior. NVIDIA-provided rootfs (Ubuntu/Yocto) support auto-insmod of all required modules.

Example Supported Auto-Insmod Modules

- > eqos_ape
- > saf775x_hwdep

- > rndis_host
- > atmel_mxt_ts
- > bluetooth
- > snd_soc_tegra_alt_t186ref_p2382
- > spidev
- > mttcan
- > bluedroid_pm
- > can_dev
- > bcmdhd
- > spi_tegra186_qspi
- > spi_tegra114
- > serial_tegra
- > pci_tegra

Modules Not Supporting Multiple Insmod

The following modules, however, do not currently support multiple insmod:

- > pci_tegra
- > bcmdhd

7.10 Crypto Interface

SE hardware module functionality is implemented in SE Server and the interface to the SE Server is implemented in the Virtual SE driver (tegra-hv-vse driver). SE hardware crypto functionality can be accessed using the standard Linux interfaces:

<https://www.kernel.org/doc/html/v4.11/crypto/>

The standard Linux kernel interface for crypto is crypto API. This includes RNG support as well. For more information, see:

<https://www.kernel.org/doc/html/v4.11/crypto/api-rng.html>

Sample code provided by the kernel to verify the crypto is available at:

<https://www.kernel.org/doc/html/v4.11/crypto/api-samples.html>

Sample code provided by the kernel to verify that RNG is available at:

<https://www.kernel.org/doc/html/v4.11/crypto/api-samples.html#code-example-for-random-number-generator-usage>



Note:

In the above example, "alg name" to be passed is "rng1-elp-tegra" or "rng1_trng" to crypto_alloc_rng for TRNG..

Refer to the SE Server documentation in the **NVIDIA DRIVE 6.0 < SDK >** for the functionality exposed by SE Server.

7.11 I2C Settings

The Linux SDK kernel supports the standard I2C kernel interfaces for specific devices, as described in:

```
<top>/drive-linux/kernel/source/oss_src/kernel/Documentation/i2c
```

This topic explains Tegra I2C settings.

Adding/Removing Devices

Tegra I2C implementation supports the standard kernel I2C interface. You can find instructions for addition, detection, and removal of I2C devices in the kernel documentation at:

```
<top>/drive-linux/kernel/source/oss_src/kernel/Documentation/i2c/writing-clients
```

7.12 PCIe Retimer

An eight-lane PCIe retimer is connected to Orin UPHY1[7:0]. It has a firmware inside, and you can upgrade this firmware using Orin I2C9.

Upgrading the PCIe Retimer Firmware

Both the firmware and upgrade tool are packaged into the file system at the following locations:

- > Firmware

```
/usr/lib/firmware/astera_retimer/firmware/pt40801/1.25.x/nvidia_p3713_x4x4/
nvidia_P3713_x4x4_X8-B2B_RETIMER-DYN_PRT_ORIENT-SRNS-CLK_IND-HOT_PLUG-
GPIO_PERST_v1_25_9.ihx
```

- > Tool

```
/lib/firmware/astera_retimer/tools/astera_ota
```

Run the following command to upgrade the retimer firmware:

```
sudo /usr/lib/firmware/astera_retimer/tools/astera_ota Install /usr/lib/firmware/
astera_retimer/firmware/pt40801/1.25.x/nvidia_p3713_x4x4/nvidia_P3713_x4x4_X8-
B2B_RETIMER-DYN_PRT_ORIENT-SRNS-CLK_IND-HOT_PLUG-GPIO_PERST_v1_25_9.ihx
```

If the flashing is successful, the following message appears:

```
nvidia@tegra-ubuntu:/usr/lib/firmware/astera_retimer/firmware/pt40801/1.25.x/
nvidia_p3713_x4x4$ sudo /usr/lib/firmware/astera_retimer/tools/astera_ota /usr/lib/
firmware/astera_retimer/firmware/pt40801/1.25.x/nvidia_p3713_x4x4/nvidia_P3713_x4x4_X8-
B2B_RETIMER-DYN_PRT_ORIENT-SRNS-CLK_IND-HOT_PLUG-GPIO_PERST_v1_25_9.ihx
```

```

15:00:53 INFO aries-sdk-c/source/aries_api.c:572: Starting Main Micro assisted EEPROM
write
15:01:19 INFO aries-sdk-c/source/aries_api.c:682: Ending Write
15:01:19 INFO aries-sdk-c/source/aries_api.c:832: Starting Main Micro assisted EEPROM
verify mode
<snip>
15:03:17 INFO aries-sdk-c/source/aries_misc.c:432:           Re-write succeeded
15:03:18 INFO aries-sdk-c/source/aries_api.c:987: End Verify
15:03:18 INFO aries-sdk-c/examples/eeprom_test.c:148: Performing PCIE HW reset ...
15:03:20 WARN aries-sdk-c/source/aries_api.c:123: No Main Micro Heartbeat
15:03:20 INFO aries-sdk-c/examples/eeprom_test.c:169: Updated FW Version is 0.0.0

```

After flashing, power cycle the Orin to take effect. You will notice the “Updated FW Version is 0.0.0” message in the log. This is a known issue and does not indicate flash failure. A solution is in progress to indicate a valid version.



Note: The tool does not prevent users from downgrading the firmware; you can flash any version of firmware following the steps.

7.13 Suspend to RAM / SC7

SC7 is an SoC power state where NVIDIA Orin[®] is in low-power mode and DRAM is in self-refresh mode. SC7 is also known as Suspend to RAM mode. In SC7, all domains in Tegra SOC will be powered down except for the AON domain. AON domain will be powered on but in sleep mode.



Note:

- > This section only talks about Suspend to Ram / SC7, which is one of many Tegra SOC power states. It does not talk about the power states at the global platform level or at the device level. You are responsible for defining power states at the platform level.
- > The term Suspend to RAM and SC7 are used interchangeably in this documentation and mean the same thing.

Legal SoC Power States

For information on the defined power states, refer to the SMCU integration guide.

Why Suspend to RAM / SC7

A significant part of the system boot time is taken up in the initialization of the application, including loading of libraries and data (DNN and map files). Achieving the boot time target in cold-boot for all use-cases is difficult.

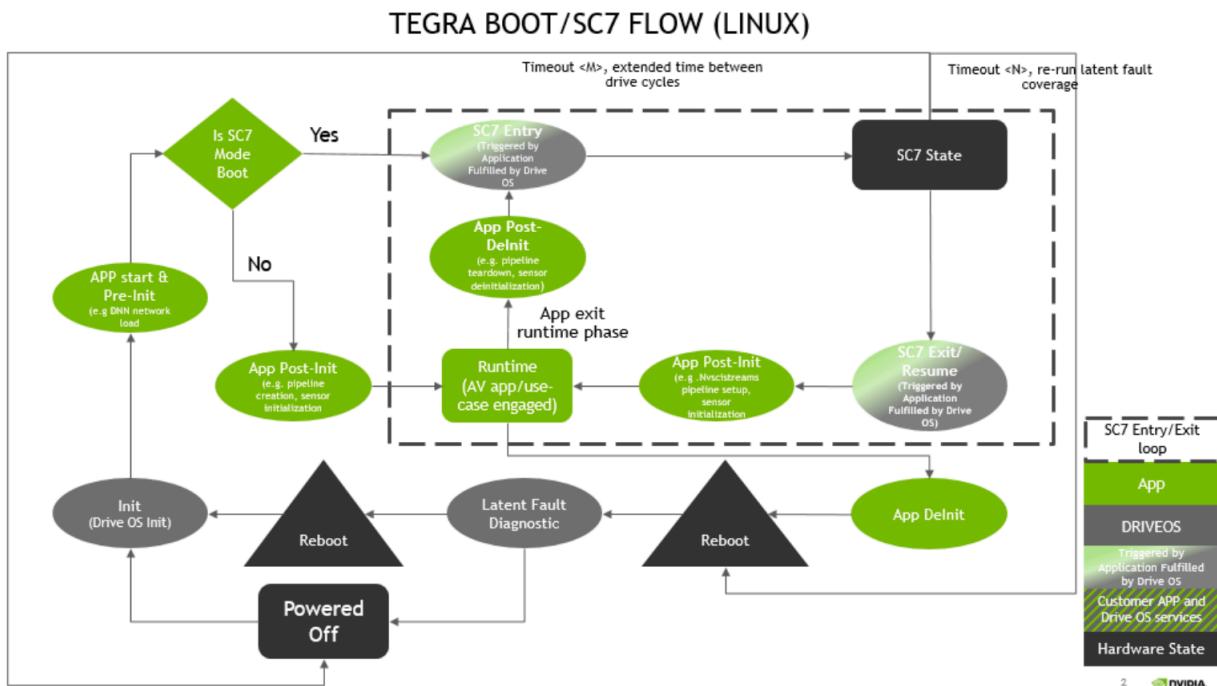
A way around this is to keep the DRAM pre-loaded and initialized before the system needs to be a fully functional state. Rather than keep the system powered off when not in use, it retains the system in SC7 state (which minimizes the power consumption while keeping the DRAM loaded and most of the software initialized).

With SC7, data is loaded and applications are initialized (example: DNN unpacking) before suspend that does not require external interaction. When it resumes, the application programs the external sensor and the rest of the application init. Since most of the initialization, which is I/O and CPU intensive, happened before suspend, the visible time is only the resume time (and much less compared to cold boot) for the end user.

How It Works

The Tegra sequence:

- When the vehicle engine is turned OFF, instead of doing a shutdown, Tegra is put in the suspend state. Optionally, before Tegra is put into suspend, KeyOFFSET can be executed.
 - Minimal power is consumed in SC7 state and requires a timeout after which you can shut down the system to save power.
 - The diagram below shows SC7 transitions and actions that need to be performed by DRIVE OS and customer applications to achieve SC7 functionality.



The App Pre-Init stage can be used for initialization and loading of libraries like CUDA and cuDNN before going into SC7 state. This can bring substantial optimizations to the Resume time, as loading of these libraries running into Gigabytes of memory can consume a lot of time. Binarized DNN can also be loaded during preinit state from memory.

The App Post-Init stage is used for initialization and allocation of resources which were not done in the Pre-Init stage. This includes powering on IO peripherals like Camera sensors, serializers-deserializers etc. The Runtime phase is meant for active processing and streaming of data captured through the various sensors. In short pipeline creation is done in this stage.

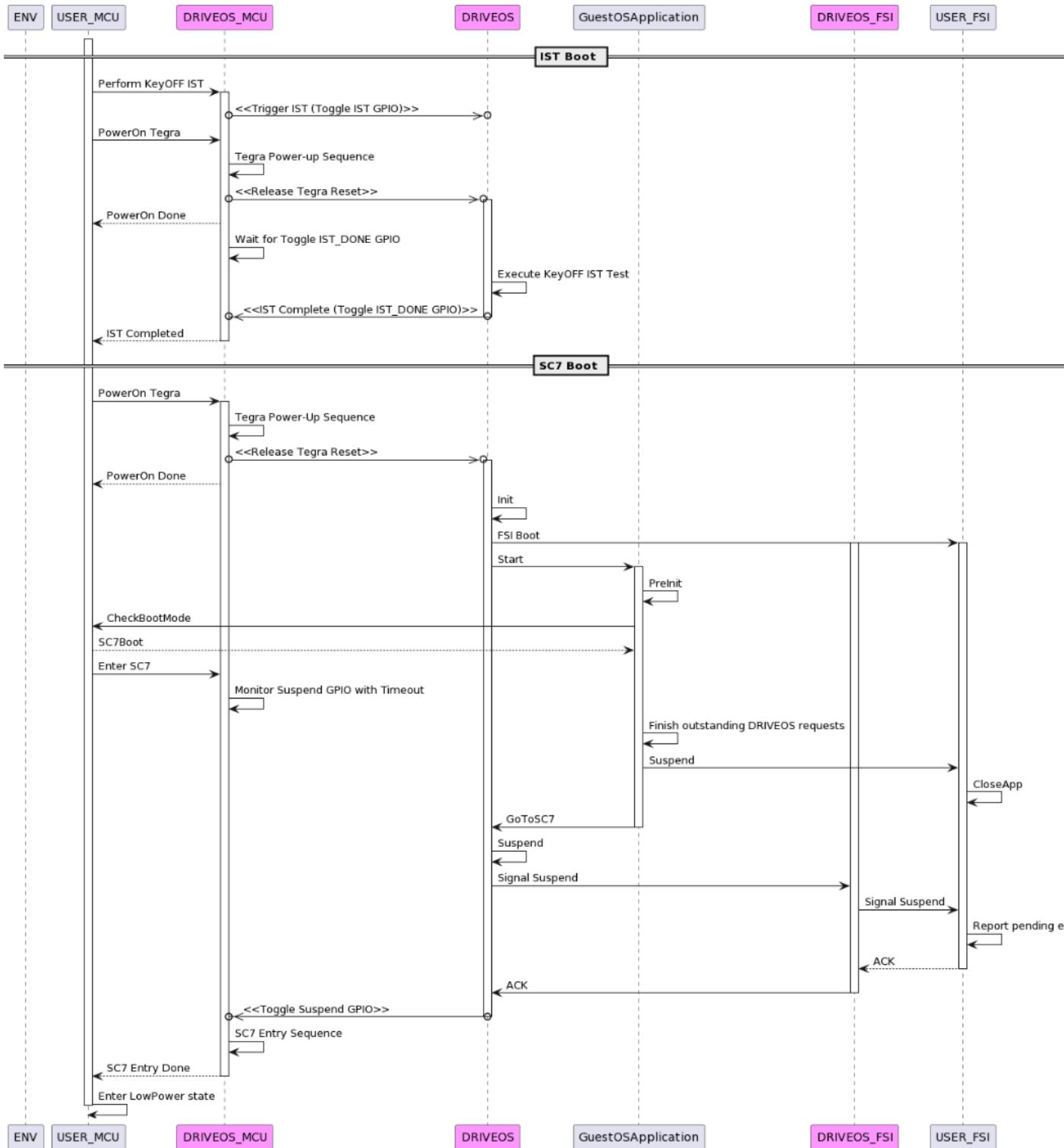
The App Post-Delnit stage is meant for freeing up resources allocated during the App Post init stage. i.e Pipeline teardown needs to be done at this stage. The loaded cuDNN and CUDA libraries shouldn't be released at this stage and these along with other resources should be freed during the App Delnit stage.

Since the Orin SoC is going to be in SC7 mode until the next vehicle key-on optionally HW latent checks can be executed once in MPFTI (every 8 hrs i.e., N=8 hours). More details are provided in the SMCU integration guide.

If the system continues to stay in SC7 for a period of M then the tegra SOC can be powered off from SMCU. The value of M needs to be decided by the customer.

SC7 Suspend Sequence

The diagram below shows the sequence to enter SC7, starting from the KeyOFF IST sequence at high level. For a detailed sequence diagram please refer to SMCU integration guide.



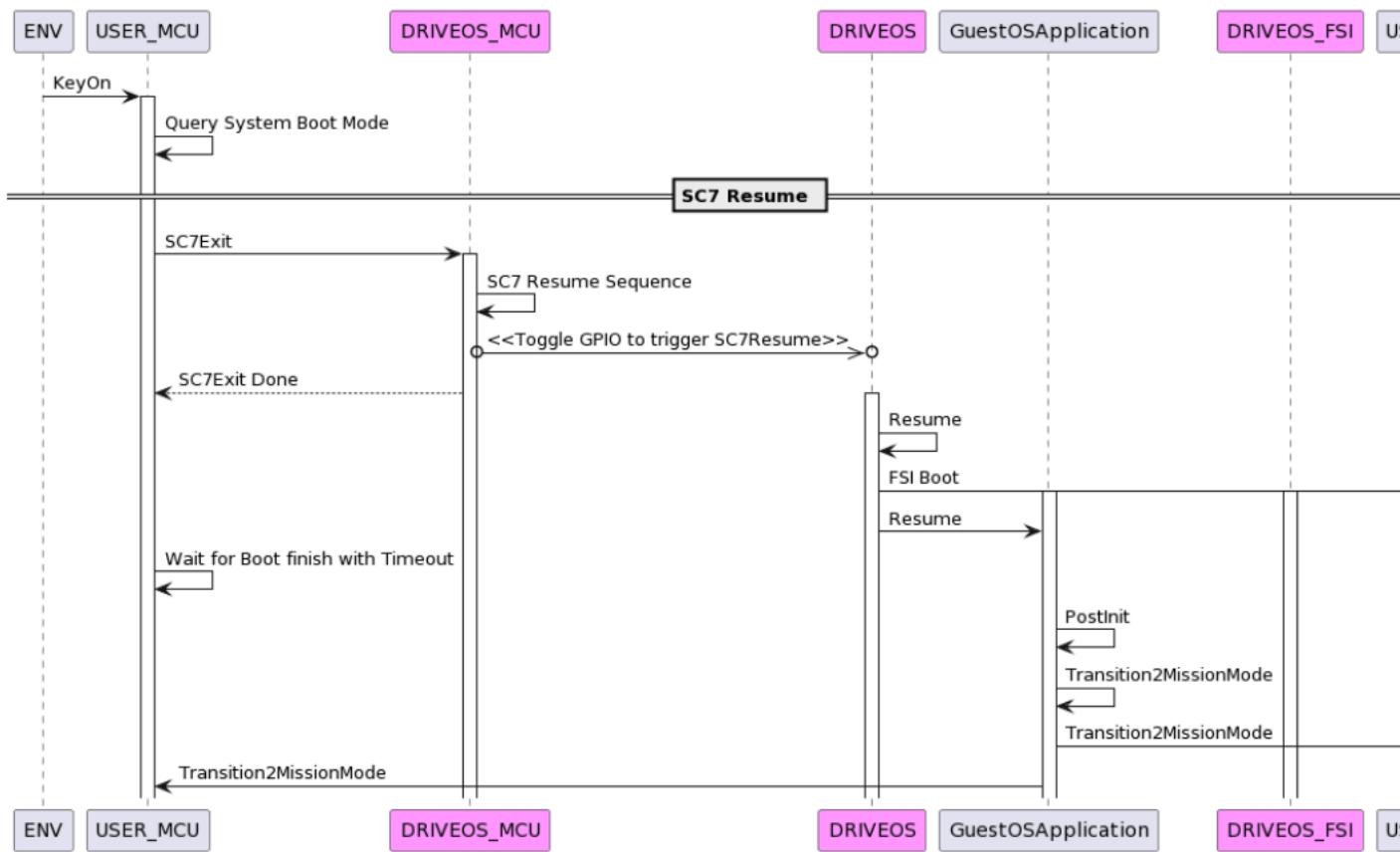
Where:

- ENV - Represents the environment that provides the event that triggers the startup or shutdown

- > DRIVEOS - Rest of the DRIVEOS software running on the Orin chip
- > DRIVEOS_FSI - FSI functionality delivered by DRIVE OS
- > DRIVEOS MCU - MCU functionality delivered by DRIVE OS
- > GuestOSApplication - User application running on the OS in a Guest Virtual Machine
- > USER_FSI - FSI firmware and user application integrated with DRIVEOS_FSI
- > USER_MCU - MCU Firmware and user application integrated with DRIVEOS_MCU

SC7 Resume Sequence

The diagram below shows the sequence to resume from SC7 state.



Interfaces Exposed by DRIVEOS MCU

Refer to the MCU Software Modules Integration Guide chapter in the *NVIDIA DRIVE OS 6.0 PDK Developer Guide* for details on Interfaces of NvMCU_SwModules.

Interface	Description
SC7Exit	Interface to wake up Tegra from suspend state.

Functionality Implemented by User Software on MCU

Functionality	Description
Transition2MissionMode	Application on Guest OS notifies the user software on MCU that the system has transitioned to mission mode.

Functionality Implemented by User Software on FSI

Table 6.

Functionality	Description
Transition2MissionMode	Application on Guest OS notifies the user software on FSI that the system has transitioned to mission mode.

Limitations

- > Camera pipeline context is not saved between suspend and resume so the camera pipeline needs to be recreated after resume and purged/teardown before suspending.
- > Any jobs submitted to any engines must be completed before SC7 is started.

Steps to Enter and Exit SC7

Prerequisite

- > Flash and boot P3710 as per SDK documentation.

Start SC7 Suspend

In Tegra shell, execute:

```
common_if_testapp -enter_sc7
echo 1 > /sys/class/tegra_hv_pm_ctl/tegra_hv_pm_ctl/device/trigger_sys_suspend
```

Exit from SC7

In AURIX shell, execute:

```
exitsc7
```

Component-Specific Limitations

- > **NvStreams:**

Before SC7 entry, ensure the following requirements for the applications using NvStreams framework to stream data between the producer and consumers:

- There are no jobs that were submitted by the NvStreams applications pending in the CPU or engine pipelines.
- The application is not performing any active NvSciSync fence waits.

To meet the requirements, perform the following tasks before proceeding to SC7 entry, if the NvStreams pipeline is being initialized:

- Wait for the buffer and synchronization primitives to be allocated and the NvSciBuf and NvSciSync objects are registered to the UMDs.
- For the NvSciStream application, wait for pipeline initialization to complete and all the packets are available in the pool block.

When the application is in the streaming phase, ensure that there are no jobs pending in the CPU or UMD engine pipelines by following the SC7 entry guidelines of the respective UMDs and that there are no packets in flight.

> **CUDA:**

Before SC7 entry, ensure the following requirements for the applications:

- Any outstanding GPU tasks (for example, kernel, memcopies, and event waits) are completed using appropriate synchronization APIs, such as `cudaEventSynchronize`, `cudaStreamSynchronize`, and `cudaDeviceSynchronize`.
- Outstanding GPU tasks can also be cross-engine dependencies (for example, between DLA and GPU) that are built using NvSciSync interop. Waiting on unsignaled NvSciSync can lead to infinite waits, so applications should ensure that the signaler of such fences has unblocked all waiters on the GPU.
- Any ongoing CUDA APIs that are synchronous with respect to the host (for example, `cudaMalloc` and `cudaStreamCreate`) are complete.
- Note that timestamps recorded by `cudaEvent` after returning from SC7 state will account for the duration in which the GPU was in a suspended state, so the elapsed time returned by `cudaEventElapsedTime` should be interpreted accordingly.

> **DLA:**

Before the SC7 entry, if the application fails to ensure that there are no outstanding tasks (submitted but not completed), the DLA detects the scenario and fails the SC7 entry.

> **PVA:**

Before SC7 entry,

- The applications must ensure that there are no pending tasks/commands on the PVA engine by waiting for all the previously submitted commands to finish before entering into SC7 by calling `cupva::Fence::wait()`.
- The applications must not submit any new commands to PVA while the DOS is transitioning to SC7 suspend state.

> **Camera:**

The SIPL camera application should only request NVIDIA DRIVE[®] OS to transition into the SC7 power state outside of the SIPL hardware initialization and SIPL runtime states.

> **NvDisplay:**

For information on the recommendations, see OpenWFD Usage Guidelines in [NvDisplay](#).

> **Graphics:**

Before SC7 entry, ensure that any tasks submitted to GPU should not be in pending state. To do so, applications must call `vkQueueWaitIdle()` for each `VkQueue` created.

> **NvScilpc:**

Before SC7 entry, ensure that there are no outstanding tasks for applications using NvScilpc. If a peer endpoint process goes down and up because it is not running during SC7 (that is, Trust Agent), applications must call `NvSciIpcResetEndpointSafe()` and then establish connection using the `NvSciIpcGetEventSafe()` API after SC7 exit.

> **NvGPU:**

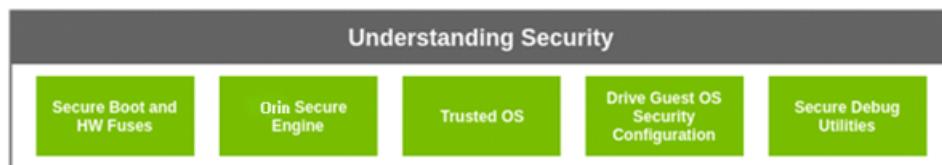
Before SC7 entry, ensure that there are no outstanding tasks for applications using NVGPU. There should not be any new request to NVGPU after the suspend callback is issued other than resume callback. Requests can be made after the driver resumes back and sets the process state to resume done.

Chapter 8. Understanding Security

NVIDIA DRIVE® OS security services ensure the confidentiality of critical system secrets such as root keys and other device configuration information. They are also responsible for providing user-space applications running in the Guest OS, the ability to offload cryptographic operations on-to SoC security hardware. These services rely on the isolation provided by the virtualization system.

This section describes the functionality and possible customization for these security services and is broadly divided into subsections.

Refer to the appropriate subsection for detailed information on the various services:



Acronyms and Abbreviations

The following acronyms are used throughout this section.

Term	Definition
ATF	ARM trusted firmware
BCT	Boot Configuration Table
BDT	Boot Device Tree
BR	BootROM
BR-BCT	BootROM Boot Configuration Table
CA	Client Applications
CBC	Cipher Block Chaining
CMAC	a block of Cipher-based Message Authentication code algorithm

Term	Definition
EKS	Encrypted Key Store
GP API	Global Platform Application Programming Interface
HW	Hardware
JTAG	Joint Test Action Group IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture
KEK	Key Encryption Key
KROM	<p>The Key ROM (KROM in short) primarily contains two types of keys:</p> <ul style="list-style-type: none"> • Wrapped Symmetric Keys • Public component of Asymmetric RSA Keys (Exponent only)
ODM	Original Design Manufacturing
OEM	Original Equipment Manufacturer
OpenSSL	A general purpose cryptography library that provides an open source implementation of the Secure Sockets Layer protocol.
OS	Operating System
OSC	Oscillator
OTA	Over-the-Air
PCT	Platform Configuration Table
PKC	Public Key Cryptography
PolarSSL	also known as ARM-mbed
REE	Rich Execution Environment
PSC	Platform Security Controller
ROM	Read-only Memory

Term	Definition
RPMB	Replay protected memory block
RSA	An encryption mechanism that uses public and private keys.
RSASSA-PSS	RSA Signature Scheme with Appendix- Probabilistic Signature Scheme (cryptography)
SBK	Secure Boot Key
SDK/PDK	Software Development Kit / Platform Development Kit
SDRAM	Synchronous Dynamic Random Access Memory
SE	Security Engine Hardware
SS	Secure Storage
TA	Trusted Applications
TEE	Trusted Execution Environment
TOS	Trusted Operating System
TSC	Tegra Secure Counter
UID	Unique Identification
UUID	Universal Unique Identification
VM	Virtual Machine

8.1 Secure Boot and Hardware Fuses

The following sections describe secure boot and hardware fuses.

8.1.1 Root of Trust and Chain of Trust

In a chain of trust, the trustworthiness of each layer of software that composes the chain is guaranteed by the previous layer, until reaching the root of the chain, or root of

trust. Immutability and formal verification provide the foundation for a root of trust. An example is the code present in read-only memory (ROM). The root of trust initiates the chain of trust.

A chain of trust, for example, can include: bootROM to boot loader to TrustZone operating system.

8.1.2 Secure Boot

Secure boot:

- > Must be implemented and enabled during manufacturing.
- > Cannot be enabled over OTA or in the field.
- > Defines a chain of trust.
- > Is supported by hardware from power on to BootROM and PSC-ROM (Platform Security Control ROM) to boot loader.
- > Must be implemented by the boot loader.

Signing Authorities

Signing authority for Orin shall consist of NVIDIA authority and OEM authority. Since the Root of Trust (ROT) resides within the silicon, NVIDIA shall enforce our authority in the Chain of Trust (COT) by baking it into the ROM. Orin shall have two ROM software running under BPMP and PSC that will establish ROT for NVIDIA. OEMs also may assert their signing authority on top of NVIDIA authority.

NVIDIA Authority

NVIDIA authority is implemented in such a way that its crypto information cannot be overwritten by the OEM. Thus, a KROM region is dedicated to keeping NVIDIA keys safe within the chip. NVIDIA keys are only readable in the encrypted form by PSC-ROM.

NVIDIA has settled on one single algorithm for each type of binary for authentication. 3072-bit RSA shall be used for all NV authentication purposes. AES-GCM (256b) shall be used for all NV authenticated encryption purposes.

- > BPMP BootROM (BPMP-BR in short)
- > PSC BootROM (PSC-ROM in short)

Both ROM software shall work with each other to establish the ROT and load/authenticate/decrypt secondary boot software (MB1) from the storage device. NVIDIA signing authority shall extend all the way to CCPLEX ARM-Core boot for T23x.

OEM Authority

OEM authority is FUSE based. FUSE is OTP entity in the chip that the OEM shall take control of during the manufacturing process of their product. OEM has the choice of enabling the different secure boot options also through the BOOT_SECURITY_INFO fuse by selecting the enablement of Secure Boot. OEM must choose a method of authentication to enable secure boot once SECURITY_MODE FUSE is burnt.

Signing Algorithms

NVIDIA authority only uses RSA3K for signing with SHA2-512 as the digest algorithm.

OEM authority has a choice of signing algorithm to choose from as stated above. The combination of signing algorithm and digest algorithm is the following:

- 3072-bit RSA (Public Exponent fixed at 0x10001) – SHA2-512 and RSAPSS encoding
- Ed25519 DSA (RFC 8032) – SHA2-512

Boot firmware signing is facilitated through Boot Component Header (BCH). The actual binaries are not signed, but their digests are included in the BCH. BCH acting like a certificate is then signed. During verification, the signature of the BCH is verified, and then the digest of the firmware within BCH is verified. Detail of BCH and verification mechanism is found in the BCH section.

For NVIDIA authority, there are signature verification public keys for each different firmware component, all the public keys reside in KROM.

For OEM authority, there are three public key hashes (PKC hash) in FUSE that the OEM can use to provision. The PKC hash is a SHA2-512 digest of a specific structure defined in SW. More detail about this is also available in the BCH section.

The FUSE_BOOT_SECURITY_INFO on NVIDIA DRIVE devices determines which sequence is used. The information in the FUSE_BOOT_SECURITY_INFO bits is as follows:

Bit[2:0]: authentication scheme:

- 001b: PKC-protected boot sequence with RSA 3K key pairs
- 100b: PKC-protected boot sequence with Ed25519 DSA key

8.1.3 Diagnostic Boot Mode

Diagnostic Boot Mode allows OEMs to bypass signing authority in later software boots on a per board basis on production mode hardware to run diagnostic software.

The flow for diagnostic boot:

- Provide ECID signed BR-BCT and MB1-BCT.
- Enable diagnostic boot in BR-BCT by setting bf_b1_diag_boot and bf_b1_skip_oem_auth_diag_boot to 1.
- Provide normal OEM signed components for MB1, MCE, PSC-BL1, and MEM-BCT.

MB1 skips OEM authentication of binaries after MCE, including the MB2 binary on the CCPLEX, which can be replaced with diagnostic software. If using standard un-signed MB2, MB2 no longer requires signed components for software it loads. In this mode, PSC-BL1, upon exit, erases all keys used for authentication and decryptions for key slots.

OEM can disable diagnostic boot mode on their devices by setting BOOT_SECURITY_INFO[10] to 1.

8.1.4 Fuse Burning Responsibilities

Applications can program a fuse by applying voltage using fuse burning routines found in the boot loader and the kernel APIs. The fuses and ownership required for Secure Boot are as follows:

NVIDIA Programmed

- > 128-bit Unique ID (UID), no two Tegra chips have the same UID.
- > NVIDIA production mode

Customer Programmed and Owned

- > FUSE_BOOT_SECURITY_INFO
- > 3 OEM Public keys hashed as SHA-512 in fuses with 2 revocable.
- > ODM fuse keys, stored in PSC fuse region.
- > ODM Production Mode (FUSE_SECURITY_MODE)



CAUTION:

ODM Production Mode fuse disables further fuse burning except field programmable fuses. Therefore, burn the ODM Production Mode fuse **last**.

8.1.4.1 Fusing a Board with a PKC Public Key Hash

Devices that implement secure boot with PKC protection have certain requirements regarding blowing fuses and boot loader signing. This topic explains how to fuse the PKC public key hash.

To fuse a board with a PKC public key hash you must have performed the following tasks:

1. Choose one of the following:

- > To generate a PKC key pair using Open SSL, or to generate signed binaries and a PKC hash, refer to [Generating a PKC Key Pair using OpenSSL](#) and the "To generate a PKC Hash" section.
- > [Fusing the Board with the Secure Keys](#)



Note:

EdDSA private key can be generated using OpenSSL Version 1.1.1.

OpenSSL Version 1.1.x changed its default digest from MD5 to SHA256.

8.1.4.2 Generating a PKC Key Pair using OpenSSL

Follow these steps to generate a PKC key pair using OpenSSL.

To install OpenSSL

1. To generate the RSA pair only, install the OpenSSL package with the following command:

```
sudo apt-get install openssl
```

To generate an EdDSA or RSA 3K key pair for NVIDIA DRIVE AGX Orin

1. Build OpenSSL as follows:

- > In a terminal window, navigate to the directory where you extracted OpenSSL and execute these commands:

```
./config  
make
```
- > When the OpenSSL build is completed, copy `libcrypto.so*` and `libssl.so*` to your local `/lib/` directory.
- > To generate the keys, execute OpenSSL from the application folder in the directory where you extracted OpenSSL.

For more information, consult the OpenSSL README file in the extracted source directory.

2. Generate EdDSA private key with the command:

```
openssl genpkey -algorithm Ed25519 --out keyfile.pem
```

3. Generate RSA 3K key pair with the command:

```
openssl genrsa -out rsa_priv.pem 3072
```

You can now generate the signed binaries and PKC hash.

To generate a PKC hash

1. Run the following commands on the host to generate a public key and a PKC hash.

- > For NVIDIA DRIVE AGX Orin™ RSA 3072-bit keys:

```
$# cd drive-foundation  
$# ./tools/flashtools/flash/tegrakeyhash --pkc  
<private_key_filename> --chip 0x23
```

Where `<private_key_filename>` depends on the tool used to generate the key.

- For PolarSSL, use `rsa_priv.txt`.
- For OpenSSL, use `rsa_priv.pem`.

- > For NVIDIA DRIVE AGX Orin EdDSA

```
$ cd drive-foundation  
$ ./tools/flashtools/flash/tegrasign_v3.py --key  
<private_key_filename> --pubkeyhash <public_key_filename> <hash_filename>
```

- Where:

`<public_key_filename>` is the name you want to give the public key file.

`<hash_filename>` is the name that you want to give the public key hash file.

`<private_key_filename>` depends on the tool used to generate the key. For OpenSSL, use `keyfile.pem`.

- Without the EdDSA private key as the input:



Note: This use case is for OEMS who want to keep the private key secure, and who only want to use the existing public key to generate the hash. The public key format should be in OpenSSL der format.

- Following is the example command to create the expected der format:

```
openssl pkey -in keyfile.pem -pubout -outform
DER > pubkey.der
```

Example Output

Following is example output of the `tegrasign_v3.py` command. The tegra-fuse format can be used in FSKP Fuse Burning Tool.

```
<fuse name="PublicKeyHash" size="64"
value="0x0123456789abcdef0123456789abcdef0123456789abcdef
0123456789abcdef0x0123456789abcdef0123456789abcdef0123456789abcdef
0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef
0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef
0123456789abcdef"/>
```

8.1.4.3 Fusing the Board with the Secure Keys

The BSP provides Fuse Burning Tool for fusing the board with the PKC hash. Sample XML files to burn secure keys and enable PKC protection are as follows.

For NVIDIA DRIVE AGX Orin RSA:

```
<genericfuse MagicId="0x45535546" version="1.0.0">
    <fuse name="PublicKeyHash" size="64" value="0x0123456789abcdef0123456789abcdef01234
56789abcdef0123456789abcdef0123456789abcdef0123456789abcdef01234
56789abcdef0123456789abcdef" />
    <fuse name="BootSecurityInfo" size="4" value="0x201" />
    <fuse name="SecurityMode" size="4" value="0x1" />
</genericfuse>
```

For NVIDIA DRIVE AGX Orin EdDSA:

```
<genericfuse MagicId="0x45535546" version="1.0.0">
    <fuse name="PublicKeyHash" size="64" value="0x0123456789abcdef0123456789abcdef01234
56789abcdef0123456789abcdef0123456789abcdef0123456789abcdef01234
56789abcdef0123456789abcdef" />
    <fuse name="BootSecurityInfo" size="4" value="0x204" />
    <fuse name="SecurityMode" size="4" value="0x1" />
</genericfuse>
```



Warning:

If used, the `SecurityMode` must be the last fuse command in the XML file. The FSKP fuse burning tool burns fuses in the order specified in the XML file. After burning the `SecurityMode` fuse, it disables fuse burning.

To set up the platform for secure boot

1. Generate the PKC Key and ensure ODM production fuse is burned.
 - [Generate a PKC Key Pair Using OpenSSL](#)
 - Ensure the ODM production fuse is burned.

2. Consult the instructions for fusing the secure keys in Factory Secure Key Provisioning.



Warning:

This step is irreversible, ensure that:

- > The PKC key pair is a valid key pair.
- > The PKC key pair is stored in a secure location, because all the binaries must be signed with the private key.

3. Sign and flash the binaries.

Use “-p <private_key_filename>” option when running the bootburn.py script.

Where <private_key_filename> depends on the tool used to generate the key:

- > For PolarSSL, use rsa_priv.txt for RSA.
- > For OpenSSL, use rsa_priv.pem for RSA, or keyfile.pem for EdDSA.

8.1.4.4 BR BCT Signing and Hash Compression

The data section in BR BCT is hashed using SHA512, and the hash digest is stored just after the 4-byte header magic at the start of BR BCT. BR uses the digest to verify BR BCT integrity before passing it to PSC-ROM for separate public-key authentication.

During boot the hash value is computed and compared with the stored value. If the update tool does not find a matching stored value, or if it finds that no value is present, the tool stores the computed value.

8.1.5 Public-Key Cryptography

Public Key Cryptography relies on a public and private key pair, allowing the manufacturer to sign a boot loader and BCT with its private key, never needing to disclose the private key at any time during the manufacturing process. The public key, which is used to verify the digital signature of the boot loader and BCT, gets stored on the secondary boot device or embedded into the boot loader and BCT images. Using PKC allows devices to be manufactured at facilities that are not secure.

8.1.5.1 Secure Boot Details with PKC Protection

The PKC-protected boot sequence is as follows. The sequence is part of the NVIDIA Secure Solutions infrastructure, which includes:

- > nvimageGen utility—manages the RSA key pairs or EdDSA key and produces secure data that includes the BCT, which can contain the BCT signatures and the public key.
- > PSC ROM—Tegra devices contain a PSC ROM that adds support for the PKC-protected boot sequence.

See the section *Signing Algorithms* under [Secure Boot](#) for details on selecting the PKC algorithm through fuse programming

PKC Secure Boot Requirements

For the PKC-secured boot process to be used, these conditions must be met. These requirements must be implemented in production because secure boot cannot be implemented using OTA onto previously non-secure boot devices.

- The fused PKC public key hash must be non-zero.
- The FUSE_BOOT_SECURITY_INFO set to the appropriate algorithm described in *Signing Algorithms* under [Secure Boot](#)
- The OEM must amend the BCT and the software components with public keys and digital signatures. Use the nvimageGen for this task.
- The ODM Production Mode, **Security Mode**, fuse must be burned and it must be the last fuse to be burned.

RSA Validation of the BCT and Boot Loader

The NVIDIA COT verification process that ensures the BCT is from the OEM, is as follows. The BootROM and PSC ROM use this sequence when the conditions in [PKC Secure Boot Requirements](#) are satisfied.



Note:

SHA-512 is the hash function used during any RSASSA-PSS operations (signature verification) in the steps below. As recommended by PKCS #1 v2.1: RSA Cryptography Standard, the manufacturer must also use SHA-512 as the same hash function applied to the message. In addition, the salt length used in the RSASSA-PSS signature verification and signature generation is the length of SHA-512 hash.

1. The BootROM reads the BCT from secondary storage.
2. The PSC ROM validates the BCT.
 - a. Validates the public key by computing SHA-512 hash and comparing it with values in fuses. If they match, the public key is stored in the PKC SE slot for BCT verification.

The public key as well as the RSASSA-PSS signature S is contained in the BCT.



Note:

The public exponent e is assumed to always be 0x10001, so it is not stored.

For NVIDIA DRIVE AGX Orin, the public key is 3072 bits.

- The PSC ROM performs a RSASSA-PSS-VERIFY signature verification operation of the BCT using the verified public key. This step validates the RSASSA-PSS signature S of the BCT. If the result of the signature verification step is a valid signature, it continues the secure boot process. If the hash comparison fails, the PSC ROM resets the system and subsequent reboot tries

the remaining 4 redundant copies of the BCT that are supported before the boot process gives up and goes to RCM.



Note:

Some copies of the BCT may have failed the public key hash compare in the beginning of Step 3, so the number of redundant copies of the BCT still available may be less than the actual number of redundant copies of the BCT written to secondary storage.

3. The BootROM reads the boot loader from secondary storage.
4. The PSC ROM validated the boot loader using the RSA public key to verify the boot loader signature.
 - a. The RSASSA-PSS signature S gets stored in the beginning of the boot loader image in the generic signature header. The PSC ROM performs a RSASSA-PSS-VERIFY signature verification operation of the boot loader. If the result of the signature verification step is a valid signature, it continues the secure boot process and the chain-of-trust is transferred to the validated boot loader. If the signature verification fails, the PSC ROM resets the system and the BootROM may attempt to load a different boot chain on reboot, depending on Boot Chain operation implemented, or go to RCM. The Boot Chain operation is described in the section [Using the Bootloader Recovery Mechanism](#).



Note:

NVIDIA DRIVE AGX Orin: the signature is verified on the image header; where the header embedded the hash value of the bootloader image.

5. The BootROM locks down security features, clears out state information, and hangs the processor.
6. The PSC ROM sets the reset vector of the BPMP processor to the next payload start vector and resets the processor to start execution of the next payload.
7. The boot loader continues the root of trust:
 - > Write protects mass storage location of the boot loader and OS.
 - > Passes execution to the validated OS image.

EdDSA Authentication of the BCT and Boot Loader

The process for EdDSA authentication is identical to the process described under the section *RSA Validation of the BCT and Boot Loader*.



Note:

NVIDIA DRIVE AGX Orin:

- > Support is provided for EdDSA operations (signature verification) using SHA-512 hash function.
- > Curve25519 is the elliptic curve supported.

Secured USB Recovery Mode

In DRIVE AGX Orin, the RCM boot flow has been merged with the normal cold boot flow. The RCM payloads are used by BootROM and PSC-ROM instead of images from secondary storage. The payload ordering and flow is described previously, in RSA Validation of the BCT and Boot Loader, for RSA and EdDSA Validation of the BCT and Boot Loader for EdDSA.

8.2 PKCS#11 Interface

The Security Services PKCS#11 library is a user space library available to DRIVE OS applications running on the Guest OS that provides a sub-set of the PKCS#11 interface as specified by the PKCS#11 v3.00 specification. In addition, some NVIDIA extensions are included.

It exposes interfaces for cryptographic hardware offload using the Security Engine for typical cryptography operations like symmetric-key/asymmetric-key cryptography, message authentication code generation, and pseudo random number generation.

Additionally, it also exposes interfaces for key management operations, including key generation, key derivation, and access to the dedicated secure key storage solution.

All cryptographic and key management operations are tightly coupled and securely implemented in SoC hardware, and the hardware-backed Trusted Execution Environment.



Note: The users of Security Services PKCS11 Lib APIs must ensure that API usage is as per API description and valid input parameters are passed. They must be familiar with the OASIS PKCS11 standard, including the OASIS standard user guide, for version 3.0, for all relevant APIs and mechanisms; they must also follow guidance published by NVIDIA in the present guide for the PKCS11 Library before using any PKCS11 API.

8.2.1 PKCS#11 – Supported Mechanism – Function Table per Token

CCPLEX Token Table (Safety and Dynamic):		
Mechanism type	Allowed operations	Allowed key type supplied to the
CKM_SHA256	CKF_DIGEST	
CKM_SHA384	CKF_DIGEST	

CCPLEX Token Table (Safety and Dynamic):

CKM_SHA512	CKF_DIGEST	
CKM_SHA3_256	CKF_DIGEST	
CKM_SHA3_384	CKF_DIGEST	
CKM_SHA3_512	CKF_DIGEST	
CKM_SHA256_HMAC	CKF_SIGN CKF_VERIFY CKF_MESSAGE_SIGN CKF_MESSAGE_VERIFY	CKK_GENERIC_
CKM_AES_CBC	CKF_ENCRYPT CKF_DECRYPT CKF_MESSAGE_ENCRYPT CKF_MESSAGE_DECRYPT CKF_WRAP CKF_UNWRAP	CKK_AES
CKM_AES_CBC_PAD	CKF_ENCRYPT CKF_DECRYPT CKF_MESSAGE_ENCRYPT CKF_MESSAGE_DECRYPT	CKK_AES
CKM_AES_CTR	CKF_ENCRYPT CKF_DECRYPT CKF_MESSAGE_ENCRYPT CKF_MESSAGE_DECRYPT	CKK_AES
CKM_AES_GCM	CKF_UNWRAP CKF_ENCRYPT CKF_DECRYPT CKF_MESSAGE_ENCRYPT CKF_MESSAGE_DECRYPT	CKK_AES
CKM_AES_CMAC	CKF_SIGN CKF_VERIFY CKF_MESSAGE_SIGN CKF_MESSAGE_VERIFY	CKK_AES
CKM_AES_GMAC	CKF_MESSAGE_SIGN CKF_MESSAGE_VERIFY	CKK_AES
CKM_RSA_PKCS_PSS	CKF_VERIFY	CKK_RSA
CKM_ECDSA	CKF_SIGN CKF_VERIFY	CKK_EC
CKM_EDDSA	CKF_SIGN CKF_VERIFY	CKK_EC_EDWA
CKM_SP800_108_COUNTER_KDF	CKF_DERIVE	CKK_AES CKK_GENERIC_

CCPLEX Token Table (Safety and Dynamic):		
CKM_ECDH1_DERIVE	CKF_DERIVE	CKK_EC CKK_EC_MONT
CKM_AES_KEY_GEN	CKF_GENERATE	
CKM_EC_EDWARDS_KEY_PAIR_GEN	CKF_GENERATE_KEY_PAIR	
CKM_EC_MONTGOMERY_KEY_PAIR_GEN	CKF_GENERATE_KEY_PAIR	
CKM_GENERIC_SECRET_KEY_GEN	CKF_GENERATE	
CKM_EC_KEY_PAIR_GEN	CKF_GENERATE_KEY_PAIR	
CKM_NVIDIA_AES_CBC_KEY_DATA_WRAP	CKF_WRAP	CKK_AES
CKM_NVIDIA_SP800_56C_TWO_STEPS_KDF	CKF_DERIVE	CKK_AES CKK_GENERIC_
CKM_NVIDIA_MACSEC_AES_KEY_WRAP	CKF_WRAP CKF_UNWRAP	CKK_AES
CKM_NVIDIA_PSC_AES_CMAC	CKF_SIGN CKF_VERIFY CKF_MESSAGE_SIGN CKF_MESSAGE_VERIFY	CKK_AES
CKM_TLS12_MASTER_KEY_DERIVE_DH	CKF_DERIVE	CKK_GENERIC_
CKM_TLS12_KDF	CKF_DERIVE	CKK_GENERIC_
CKM_TLS12_MAC	CKF_SIGN CKF_VERIFY	CKK_GENERIC_
CKM_TLS12_KEY_AND_MAC_DERIVE	CKF_DERIVE	CKK_GENERIC_
CKM_TLS12_KEY_SAFE_DERIVE	CKF_DERIVE	CKK_GENERIC_
CKM_NVIDIA_AES_GCM_KEY_UNWRAP	CKF_UNWRAP	CKK_AES

TSEC Dynamic Token Table

Mechanism type	Allowed operations	Allowed key type supplied to the
CKM_SP800_108_COUNTER_KDF	CKF_DERIVE	CKK_AES CKK_GENERIC_

TSEC Safety Token Table

Mechanism type	Allowed operations	Allowed key type supplied to the
CKM_AES_CMAC	CKF_SIGN CKF_VERIFY	CKK_AES

FSI Dynamic Token Table

Mechanism type	Allowed operations	Allowed key type supplied to the
CKM_SP800_108_COUNTER_KDF	CKF_DERIVE	CKK_AES CKK_GENERIC_
CKM_AES_KEY_GEN	CKF_GENERATE	
CKM_GENERIC_SECRET_KEY_GEN	CKF_GENERATE	
CKM_AES_GCM	CKF_UNWRAP	CKK_AES
CKM_NVIDIA_AES_GCM_KEY_UNWRAP	CKF_UNWRAP	CKK_AES
CKM_EC_EDWARDS_KEY_PAIR_GEN	CKF_GENERATE_KEY_PAIR	
CKM_EC_MONTGOMERY_KEY_PAIR_GEN	CKF_GENERATE_KEY_PAIR	
CKM_EC_KEY_PAIR_GEN	CKF_GENERATE_KEY_PAIR	
CKM_ECDH1_DERIVE	CKF_DERIVE	CKK_EC CKK_EC_MONT

8.2.2 PKCS#11 – Supported APIs

The following table describes supported APIs:

Category	API from PKCS#11 v3.0
General	C_Initialize
	C_Finalize
	C_GetInfo
	C_GetFunctionList
	C_GetInterfaceList
	C_GetInterface
Token	C_GetSlotList
	C_GetSlotInfo
	C_GetTokenInfo
	C_GetMechanismList
	C_GetMechanismInfo
Session	C_OpenSession
	C_CloseSession
	C_CloseAllSessions
	C_GetSessionInfo
	C_SessionCancel
	C_Login
	C_Logout
Object	C_CreateObject
	C_CopyObject
	C_DestroyObject
	C_GetAttributeValue
	C_SetAttributeValue

Category	API from PKCS#11 v3.0
Symmetric Encrypt and Decrypt	C_FindObjectsInit
	C_FindObjects
	C_FindObjectsFinal
	C_EncryptInit
	C_Encrypt
	C_EncryptUpdate
	C_EncryptFinal
	C_DecryptInit
	C_Decrypt
	C_DecryptUpdate
	C_DecryptFinal
Digest	C_DigestInit
	C_Digest
	C_DigestUpdate
	C_DigestFinal
Sign and Verify Signatures and MACs	C_SignInit
	C_Sign
	C_VerifyInit
	C_Verify
Key Operations	C_GenerateKey
	C_GenerateKeyPair
	C_WrapKey
	C_UnwrapKey
	C_DeriveKey
RNG	C_GenerateRandom

Category	API from PKCS#11 v3.0
Message-based Sign and Verify signatures and MACs	C_MessageSignInit C_SignMessage C_SignMessageBegin C_SignMessageNext C_MessageSignFinal C_MessageVerifyInit C_VerifyMessage C_VerifyMessageBegin C_VerifyMessageNext C_MessageVerifyFinal
Symmetric Message-based Encrypt and Decrypt	C_MessageEncryptInit C_EncryptMessage C_EncryptMessageBegin C_EncryptMessageNext C_MessageEncryptFinal C_MessageDecryptInit C_DecryptMessage C_DecryptMessageBegin C_DecryptMessageNext C_MessageDecryptFinal

NVIDIA Extensions

The following table shows NVIDIA API extensions:

NVIDIA Extensions: API
C_NVIDIA_EncryptGetIV
C_NVIDIA_CommitTokenObjects

NVIDIA Extensions: API

C_NVIDIA_InitializeChannel

C_NVIDIA_OpenSession

C_NVIDIA_FinalizeChannel

C_NVIDIA_SetKATParameters

NVIDIA Extensions: Error Return

CKR_NVIDIA_SECURE_STORAGE_FAILED

CKR_NVIDIA_SECURE_STORAGE_TAMPERED

CKR_NVIDIA_CHANNEL_NOT_FOUND

CKR_NVIDIA_CHANNEL_CANNOT_OPEN

CKR_NVIDIA_OBJECTS_CHANGED

8.2.3 PKCS#11 – Supported Objects

PKCS#11 library supports the following objects:

- Public key (CKO_PUBLIC_KEY)
- Private key (CKO_PRIVATE_KEY)
- Secret key (CKO_SECRET_KEY)
- Data Object (CKO_DATA)

8.2.4 PKCS#11 – Persistent Object Secure Storage Support

The following APIs can operate on the objects in both token (persistent) and session (ephemeral) mode if the token secure storage is available.

- C_CopyObject
- C_DestroyObject
- C_SetAttributeValue
- C_GenerateKey
- C_UnwrapKey
- C_WrapKey
- C_DeriveKey
- C_CreateObject

Token Storage Status

The status of a token's secure storage and the status of a token itself can be established by calling C_GetTokenInfo.

Token Information flags have been extended in the PKCS#11 library implementation. These follow on from “CKF_ERROR_STATE” defined in Table 6 of PKCS#11 v3.00 specification.

NVIDIA Extensions: Token Information Flags
CKF_NVIDIA_TOKEN_OK
CKF_NVIDIA_SECURE_STORAGE_FAILED
CKF_NVIDIA_SECURE_STORAGE_TAMPERED
CKF_NVIDIA_KEYLOAD_TIMEOUT
CKF_NVIDIA_KEYLOAD_FAILED
CKF_NVIDIA_TOKEN_ERROR
CKF_NVIDIA_SECURE_STORAGE_NOT_PROVISIONED
CKF_NVIDIA_SECURE_STORAGE_NOT_PRESENT

The PKCS#11 Library CK_TOKEN_INFO structure contains the following values:

ulMaxSessionCount	PKCS#11 Specification: maximum number of sessions that can be opened with the token at one time by a single application.	NVIDIA Implementation: represents the total number of sessions available to a library instance across all tokens.
ulMaxRwSessionCount	PKCS#11 Specification: Maximum number of read/write sessions that can be opened with the token at one time by a single application.	NVIDIA Implementation: When both the token and token secure storage status are OK, it represents the total number of read/write sessions available to a library instance across all tokens; otherwise, it will remain as CK_UNAVAILABLE_INFORMATION.

To confirm the status, the application recommended sequence is:

- C_Initialize()
- C_GetSlotList(), and then find the slot/token you require
- C_GetTokenInfo(), and then check the flags entry for CKF_NVIDIA_TOKEN_OK

8.2.5 PKCS#11 – Supported Attributes

Create EC and RSA Public Key Attributes Support

The following table lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type being created.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for the specific key type.
No	Indicates that PKCS#11 library does not support the attribute for the specific key type.
Read-only	The attribute is set to read-only for the specific key type.
	An empty cell in Default Value column indicates no specific value is assigned to the attribute.
(Result of library function)	Indicates that the attribute value is determined by the PKCS#11 library.

Attributes	KeyTypes		Default Values	Note
	EC Public	RSA Public		
CKA_CLASS	Yes	Yes	CKO_PUBLIC_KEY	Mandatory template attribute.
CKA_TOKEN	Yes	Yes	FALSE	-
CKA_PRIVATE	Read-only	Read-only	TRUE	NVIDIA limitation. All objects are private.
CKA_LABEL	Yes	Yes		
CKA_VALUE	No	No		
CKA_TRUSTED	Read-only	Read-only	FALSE	NVIDIA limitation. Cannot create a trusted wrapping key at runtime.

C_CreateObject				
Attributes	KeyTypes		Default Values	Note
	EC	RSA	Public	
CKA_CHECK_VALUE	No	No		
CKA_KEY_TYPE	Yes	Yes		Mandatory template attribute.
CKA SUBJECT	No	No		NVIDIA limitation. Attribute not supported.
CKA_ID	Yes	Yes		Mandatory template .
CKA_SENSITIVE	No	No		
CKA_ENCRYPT	Read-only	Read-only	FALSE	NVIDIA limitation. Public key encryption is not supported.
CKA_DECRYPT	No	No		
CKA_WRAP	Read-only	Read-only	FALSE	NVIDIA limitation. Public key wrap is not supported.
CKA_UNWRAP	No	No		
CKA_SIGN	No	No		
CKA_VERIFY	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules
CKA_VERIFY_RECOVER	No	No		NVIDIA limitation. Attribute not supported.
CKA_DERIVE	Read-only	Read-only	FALSE	NVIDIA limitation. Cannot derive from a Public key.

C_CreateObject				
Attributes	KeyTypes		Default Values	Note
	EC	RSA	Public	
CKA_START_DATE	Yes	Yes		
CKA_END_DATE	Yes	Yes		
CKA_MODULUS	No	Yes		Mandatory template attribute.
CKA_MODULUS_BITS	No	Read-only	(Result of library function)	Must not be template attribute.
CKA_PUBLIC_EXPONENT	No	Yes		Mandatory template attribute.
CKA_PUBLIC_KEY_INFO	No	No		NVIDIA limitation. Attribute not supported.
CKA_VALUE_LEN	No	No		
CKA_EXTRACTABLE	No	No		
CKA_LOCAL	Read-only	Read-only	FALSE	Must not be template attribute.
CKA_NEVER_EXTRACTABLE	No	No		
CKA_ALWAYS_SENSITIVE	No	No		
CKA_KEY_GEN_MECHANISM	Read-only	Read-only	CK_UNAVAILABLE_INFORMATION	Due to CKA_LOCAL set FALSE.
CKA_MODIFIABLE	Yes	Yes	TRUE	
CKA_COPYABLE	Yes	Yes	TRUE	
CKA_DESTROYABLE	Yes	Yes	TRUE	

C_CreateObject				
Attributes	KeyTypes		Default Values	Note
	EC	RSA	Public	
CKA_EC_PARAMS	Yes	No		Mandatory template attribute.
CKA_EC_POINT	Yes	No		Mandatory template attribute.
CKA_WRAP_WITH_TRUSTED	No	No		
CKA_WRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.
CKA_UNWRAP_TEMPLATE	No	No		
CKA_ALLOWED_MECHANISM	Yes	Yes		Mandatory template attribute.
CKA_NVIDIA_CALLER_NONCE	No	No		

Create Secret Key Attributes Support

The table below lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type being created.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for the specific key type.
No	Indicates that PKCS#11 library does not support the attribute for the specific key type.
Read-only	The attribute is set to read-only for the specific key type.
	An empty cell in Default Value column indicates there is no specific value assigned to the attribute.
(Result of library function)	Indicates that the PKCS#11 library determines the attribute value.

C_CreateObject				
Attributes	Key Type		Default Value	Note
	Generic Secret	AES		
CKA_CLASS	Yes	Yes	CKO_SECRET_KEY	Mandatory template attribute.
CKA_TOKEN	Yes	Yes	FALSE	
CKA_PRIVATE	Read-only	Read-only	TRUE	NVIDIA limitation. All objects are private.
CKA_LABEL	Yes	Yes		
CKA_VALUE	Yes	Yes		Mandatory template attribute.
CKA_TRUSTED	Read-only	Read-only	FALSE	NVIDIA limitation. Cannot create a trusted wrapping key at runtime.
CKA_CHECK_VALUE	No	No		NVIDIA limitation. Attribute not supported.
CKA_KEY_TYPE	Yes	Yes		Mandatory template attribute.
CKA_SUBJECT	No	No		NVIDIA limitation. Attribute not supported.
CKA_ID	Yes	Yes		Mandatory template attribute.
CKA_SENSITIVE	Read-only	Read-only	TRUE	NVIDIA limitation. No access to secret key material.
CKA_ENCRYPT	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.

C_CreateObject				
Attributes	Key Type		Default Value	Note
	Generic Secret	AES		
CKA_DECRYPT	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_WRAP	No	Yes	FALSE	
CKA_UNWRAP	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_SIGN	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY_RECOV	No	No		
CKA_DERIVE	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_START_DATE	Yes	Yes		
CKA_END_DATE	Yes	Yes		
CKA_MODULUS	No	No		
CKA_MODULUS_BIT	No	No		
CKA_PUBLIC_EXPON	No	No		
CKA_PUBLIC_KEY_IN	No	No		
CKA_VALUE_LEN	Read-only	Read-only	(Result of library function)	Must not be template attribute.
CKA_EXTRACTABLE	Yes	Yes	FALSE	
CKA_LOCAL	Read-only	Read-only	FALSE	Must not be template attribute.

C_CreateObject				
Attributes	Key Type		Default Value	Note
	Generic Secret	AES		
CKA_NEVER_EXTRACTABLE	Read-only	Read-only	FALSE	Must not be template attribute.
CKA_ALWAYS_SENSITIVE	Read-only	Read-only	FALSE	Must not be template attribute.
CKA_KEY_GEN_MECHANISM	Read-only	Read-only	CK_UNAVAILABLE_INFORMATION	Due to CKA_LOCAL set FALSE.
CKA_MODIFIABLE	Yes	Yes	TRUE	
CKA_COPYABLE	Yes	Yes	TRUE	
CKA_DESTROYABLE	Yes	Yes	TRUE	
CKA_EC_PARAMS	No	No		
CKA_EC_POINT	No	No		
CKA_WRAP_WITH_TEMPLATE	Yes	Yes	FALSE	
CKA_WRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported
CKA_UNWRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.
CKA_ALLOWED_MECHANISMS	Yes	Yes		Mandatory template attribute.
CKA_NVIDIA_CALLED	Read-only	Read-only	FALSE	

Generate Secret Key Attributes Support

The following table lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type being generated.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for the specific key type.

Table Entry	Meaning
No	Indicates that PKCS#11 library does not support the attribute for the specific key type.
Read-only	The attribute is set to read-only for the specific key type.
	An empty cell in the Default Value column indicates there is no specific value assigned to the attribute.
(Result of library function)	Indicates that the attribute value is determined by the PKCS#11 library.

C_GenerateKey				
Attributes	Key Type		Default Value	Note
	Generic Secret	AES		
CKA_CLASS	Read-only	Read-only	CKO_SECRET_KEY	Implied by generation mechanism. Cannot be changed.
CKA_TOKEN	Yes	Yes	FALSE	
CKA_PRIVATE	Read-only	Read-only	TRUE	NVIDIA limitation. All objects are private.
CKA_LABEL	Yes	Yes		
CKA_VALUE	Read-only	Read-only	(Result of library function)	Is set by mechanism.
CKA_TRUSTED	Read-only	Read-only	FALSE	NVIDIA limitation. Cannot create a trusted wrapping key at runtime.
CKA_CHECK_VALUE	No	No		NVIDIA limitation. Attribute not supported.

C_GenerateKey				
Attributes	Key Type		Default Value	Note
CKA_KEY_TYPE	Read-only	Read-only	(Result of library function)	Is set by mechanism Cannot be changed.
CKA SUBJECT	No	No		
CKA_ID	Yes	Yes		Mandatory template attribute.
CKA_SENSITIVE	Read-only	Read-only	TRUE	NVIDIA limitation. No access to Secret key material.
CKA_ENCRYPT	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_DECRYPT	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_WRAP	No	Yes	FALSE	
CKA_UNWRAP	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_SIGN	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY_REC	No	No		

C_GenerateKey				
Attributes	Key Type		Default Value	Note
CKA_DERIVE	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_START_DATE	Yes	Yes		
CKA_END_DATE	Yes	Yes		
CKA_MODULUS	No	No		
CKA_MODULUS_BIT	No	No		
CKA_PUBLIC_EXPON	No	No		
CKA_PUBLIC_KEY_IN	No	No		
CKA_VALUE_LEN	Yes	Yes	16	Mandatory template attribute.
CKA_EXTRACTABLE	Yes	Yes	FALSE	
CKA_LOCAL	Read-only	Read-only	TRUE	Must not be template attribute.
CKA_NEVER_EXTRACTABLE	Read-only	Read-only	(Result of library function)	Must not be template attribute.
CKA_ALWAYS_SENSITIVE	Read-only	Read-only	TRUE	Must not be template attribute. NVIDIA limitation. No access to Secret key material.
CKA_KEY_GEN_MEC	Read-only	Read-only	(Result of library function)	Must not be template attribute.

C_GenerateKey				
Attributes	Key Type		Default Value	Note
CKA_MODIFIABLE	Yes	Yes	TRUE	
CKA_COPYABLE	Yes	Yes	TRUE	
CKA_DESTROYABLE	Yes	Yes	TRUE	
CKA_EC_PARAMS	No	No		
CKA_EC_POINT	No	No		
CKA_WRAP_WITH_TL	Yes	Yes	FALSE	
CKA_WRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.
CKA_UNWRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.
CKA_ALLOWED_MECHANISM	Yes	Yes		Mandatory template attribute.
CKA_NVIDIA_CALLEE	Read-only	Read-only	FALSE	

Generate Public / Private Key Pair Attributes Support

The following table lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type being generated.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for the specific key type.
No	Indicates that PKCS#11 library does not support the attribute for the specific key type.

Table Entry	Meaning
Read-only	The attribute is set to read-only for the specific key type.
	An empty cell in Default Value column indicates there is no specific value assigned to the attribute.
(Result of library function)	Indicates that the PKCS#11 library determines the attribute value.

C_GenerateKeyPair				
Attributes	Key Type		Default Value	Note
	EC Public	EC Private		
CKA_CLASS	Read-only	Read-only	(Result of library function)	
CKA_TOKEN	Yes	Yes	FALSE	Same value for both templates.
CKA_PRIVATE	Read-only	Read-only	TRUE	NVIDIA limitation. All objects are private.
CKA_LABEL	Yes	Yes		
CKA_VALUE	No	No		
CKA_TRUSTED	Read-only	No	FALSE	NVIDIA limitation. Cannot create a trusted wrapping key at runtime.
CKA_CHECK_VALUE	No	No		
CKA_KEY_TYPE	Read-only	Read-only	(Result of library function)	
CKA_SUBJECT	No	No		NVIDIA limitation. Attribute not supported.
CKA_ID	Yes	Yes		Mandatory template attribute, they must be identical.

C_GenerateKeyPair				
Attributes	Key Type		Default Value	Note
	EC Public	EC Private		
CKA_SENSITIVE	No	Read-only	TRUE	NVIDIA limitation. No access to private key material.
CKA_ENCRYPT	Read-only	No	FALSE	NVIDIA limitation. Public key encryption is not supported.
CKA_DECRYPT	No	Read-only	FALSE	NVIDIA limitation. Private key decryption is not supported.
CKA_WRAP	Read-only	No	FALSE	NVIDIA limitation. Public key wrap is not supported.
CKA_UNWRAP	No	Read-only	FALSE	NVIDIA limitation. Private key unwrap is not supported.
CKA_SIGN	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_SIGN_RECOVER	No	No	-	NVIDIA limitation. Attribute not supported.
CKA_VERIFY	Yes	No	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY_RECOV	No	No	-	NVIDIA limitation. Attribute not supported.
CKA_DERIVE	Read-only	Yes	FALSE	NVIDIA limitation. Cannot derive from a public key.
CKA_START_DATE	Yes	Yes		
CKA_END_DATE	Yes	Yes		

C_GenerateKeyPair

Attributes	Key Type		Default Value	Note
	EC Public	EC Private		
CKA_MODULUS	No	No		
CKA_MODULUS_BIT	No	No		
CKA_PUBLIC_EXPON	No	No		
CKA_PUBLIC_KEY_IN	No	No		NVIDIA limitation. Attribute not supported.
CKA_VALUE_LEN	No	No		
CKA_EXTRACTABLE	No	Yes	FALSE	
CKA_LOCAL	Read-only	Read-only	TRUE	Must not be template attribute.
CKA_NEVER_EXTRACTABLE	No	Read-only	(Result of library function)	Must not be template attribute
CKA_ALWAYS_SENSITIVE	No	Read-only	TRUE	Must not be template attribute. NVIDIA limitation. No access to private key material.
CKA_KEY_GEN_MEC	Read-only	Read-only	(Result of library function)	Must not be template attribute.
CKA_MODIFIABLE	Yes	Yes	TRUE	
CKA_COPYABLE	Yes	Yes	TRUE	
CKA_DESTROYABLE	Yes	Yes	TRUE	
CKA_EC_PARAMS	Yes	Read-only		Public key: mandatory template attribute. Private key: must not be template attribute.

C_GenerateKeyPair

Attributes	Key Type		Default Value	Note
	EC Public	EC Private		
CKA_EC_POINT	Read-only	No	(Result of library function)	
CKA_WRAP_WITH_TL	No	Yes	FALSE	
CKA_WRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.
CKA_UNWRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.
CKA_ALLOWED_MECHANISMS	Yes	Yes		Mandatory template attribute.
CKA_ALWAYS_AUTH	No	No		NVIDIA limitation. Not supported for private keys.
CKA_NVIDIA_CALLED	No	No		

Derive Secret Key Attributes Support

The table below lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type being derived.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for the specific key type.
No	Indicates that PKCS#11 library does not support the attribute for the specific key type.
Read-only	The attribute is set to read-only for the specific key type.
	An empty cell in Default Value column indicates there is no specific value assigned to the attribute.
(Result of library function)	Indicates that the PKCS#11 library determines the attribute value.

C_DeriveKey				
Attributes	Key Type		Default Value	Note
	Gene Secret AES			
CKA_CLASS	Read-only	Read-only	CKO_SECRET_KEY	NVIDIA limitation. Can only derive a Secret key.
CKA_TOKEN	Yes	Yes	FALSE	NVIDIA limitation. Can only derive a Token key from a Token key.
CKA_PRIVATE	Read-only	Read-only	TRUE	NVIDIA limitation. All objects are private.
CKA_LABEL	Yes	Yes		
CKA_VALUE	Read-only	Read-only	(Result of library function)	
CKA_TRUSTED	Read-only	Read-only	FALSE	NVIDIA limitation. Cannot create a trusted wrapping key at runtime.
CKA_CHECK_VALUE	No	No		NVIDIA limitation. Not supported.
CKA_KEY_TYPE	Yes	Yes		Mandatory template attribute.
CKA_SUBJECT	No	No		
CKA_ID	Yes	Yes		Mandatory template attribute.
CKA_SENSITIVE	Yes	Yes	TRUE	NVIDIA limitation. Any Secret Key with CKA_SENSITIVE False cannot be used for cryptographic operations.
CKA_ENCRYPT	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.

C_DeriveKey				
Attributes	Key Type		Default Value	Note
	Gene	Secret	AES	
CKA_DECRYPT	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_WRAP	No	Yes	FALSE	
CKA_UNWRAP	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_SIGN	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY_RECOVER	No	No		
CKA_DERIVE	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_START_DATE	Yes	Yes		
CKA_END_DATE	Yes	Yes		
CKA_MODULUS	No	No		
CKA_MODULUS_BITS	No	No		
CKA_PUBLIC_EXPONENT	No	No		
CKA_PUBLIC_KEY_INFO	No	No		
CKA_VALUE_LEN	Yes	Yes	16	Mandatory template attribute.

C_DeriveKey				
Attributes	Key Type		Default Value	Note
	Gene	Secret	AES	
CKA_EXTRACTABLE	Yes	Yes	FALSE	NVIDIA limitation. If the base key has its CKA_EXTRACTABLE attribute set to CK_FALSE, then the derived key will too
CKA_LOCAL	Read-only	Read-only	FALSE	Must not be template attribute.
CKA_NEVER_EXTRACTABLE	Read-only	Read-only	Inherited from base key depending on CKA_EXTRACTABLE history*	Must not be template attribute.
CKA_ALWAYS_SENSITIVE	Read-only	Read-only	Inherited from base key depending on CKA_SENSITIVE history**	Must not be template attribute.
CKA_KEY_GEN_MECHANISM	Read-only	Read-only	CK_UNAVAILABLE_INFORMATION	Due to CKA_LOCAL set FALSE
CKA_MODIFIABLE	Yes	Yes	TRUE	
CKA_COPYABLE	Yes	Yes	TRUE	
CKA_DESTROYABLE	Yes	Yes	TRUE	
CKA_EC_PARAMS	No	No		
CKA_EC_POINT	No	No		
CKA_WRAP_WITH_TRUSTED	Yes	Yes	FALSE	
CKA_WRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.

C_DeriveKey				
Attributes	Key Type		Default Value	Note
	Gene	Secret	AES	
CKA_UNWRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.
CKA_ALLOWED_MECHANISM	Yes	Yes		Mandatory template attribute
CKA_NVIDIA_CALLER_NONCE	Yes	Yes	FALSE	NVIDIA Extension May be TRUE only for encrypt/decrypt session keys derived using CKM_TLS12_KEY_AND_MAC_DERIVE or CKM_TLS12_KEY_SAFE_DERIVE

* If the base key has its CKA_NEVER_EXTRACTABLE attribute set to CK_FALSE, then the derived key will too. If the base key has its CKA_NEVER_EXTRACTABLE attribute set to CK_TRUE, then the derived key has its CKA_NEVER_EXTRACTABLE attribute set to the opposite value from its CKA_EXTRACTABLE attribute. If the base key has its CKA_EXTRACTABLE attribute set to CK_FALSE, then the derived key will too.

** If the base key has its CKA_ALWAYS_SENSITIVE attribute set to CK_FALSE, then the derived key will as well. If the base key has its CKA_ALWAYS_SENSITIVE attribute set to CK_TRUE, then the derived key has its CKA_ALWAYS_SENSITIVE attribute set to the same value as its CKA_SENSITIVE attribute.

Unwrap Key Attributes Support with CKM_NVIDIA_AES_GCM_KEY_UNWRAP

PKCS#11 library does not support Cryptoki attributes supplied within a template to be applied to the unwrapped key with CKM_AES_GCM mechanism. The key attributes are instead supplied via the optional Additional authenticated Data (AAD) input when CKM_NVIDIA_AES_GCM_KEY_UNWRAP mechanism is called with C_UnwrapKey.

This change uses a vendor-specific mechanism introduced at 6.0.8.1. It is backwards compatible for customers who already created unwrapping keys with CKM_AES_GCM as the supported mechanism, and where both CKM_AES_GCM and CKM_NVIDIA_AES_GCM_KEY_UNWRAP can coexist and be used.

How does this change affect the customer application? Depending on the combination of how the supported mechanism in the unwrapping key is named, and how the unwrapping mechanism is named when C_UnwrapKey is called, the PKCS#11 library reacts according to one of the four possible combinations, as shown below:

Unwrap Key Mechanism			
		CKM_AES_GCM	CKM_NVIDIA_AES_GCM_KEY_UNWRAP
Supported mechanism in the unwrapping key	CKM_AES_GCM	<ul style="list-style-type: none"> > If pTemplate == NULL unwrap using CKM_NVIDIA_AES Log ("Update unwrapping mechanism and key supported mechanisms") > If pTemplate != NULL return CKR_ARGUMENTS_BAD (not supported as of 6.0.8.1) 	<ul style="list-style-type: none"> > If pTemplate == NULL unwrap using CKM_NVIDIA_AES_GCM_KEY_U Log("Update key supported mechanisms") > If pTemplate != NULL return CKR_ARGUMENTS_BAD
	CKM_NVIDIA_AES_GCM_KEY_UNWRAP	<ul style="list-style-type: none"> return CKR_MECHANISM_INVALID 	<ul style="list-style-type: none"> > If pTemplate == NULL unwrap using CKM_NVIDIA_AES_GCM_KEY_U > If pTemplate != NULL return CKR_ARGUMENTS_BAD

A customer application provisioning keys using the original mechanism will still work with 6.0.8.1. The PKCS#11 Library issues an advisory log to update to the new vendor-specific mechanism naming scheme for that use case.

Unwrap Secret Key Attributes Support with CKM_AES_CBC

PKCS#11 library does support Cryptoki attributes supplied within a template to be applied to the unwrapped key with CKM_AES_CBC mechanism.

The following table lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type being created.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for the specific key type.
No	Indicates that PKCS#11 library does not support the attribute for the specific key type.
Read-only	The attribute is set to read-only for the specific key type.
	An empty cell in Default Value column indicates that there is no specific value assigned to the attribute.
(Result of library function)	Indicates that the attribute value is determined by the PKCS#11 library

C_UnwrapKey				
Attributes	Key Type		Default Value	Note
	Generic Secret	AES		
CKA_CLASS	Yes	Yes	CKO_SECRET_KEY	Mandatory template attribute.
CKA_TOKEN	Read-only	Read-only	FALSE	NVIDIA limitation. Only EPHEMERAL keys can be unwrapped if attribute template is supported.
CKA_PRIVATE	Read-only	Read-only	TRUE	NVIDIA limitation. All objects are private.
CKA_LABEL	Yes	Yes		
CKA_VALUE	No	No		
CKA_TRUSTED	Read-only	Read-only	FALSE	NVIDIA limitation. Cannot create a trusted wrapping key at runtime.

C_UnwrapKey				
Attributes	Key Type		Default Value	Note
	Generic Secret	AES		
CKA_CHECK_VALUE	No	No		NVIDIA limitation. Attribute not supported.
CKA_KEY_TYPE	Yes	Yes		Mandatory template attribute.
CKA SUBJECT	No	No		NVIDIA limitation. Attribute not supported.
CKA_ID	Yes	Yes		Mandatory template attribute.
CKA_SENSITIVE	Read-only	Read-only	TRUE	NVIDIA limitation. No access to secret key material.
CKA_ENCRYPT	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_DECRYPT	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_WRAP	No	Yes	FALSE	
CKA_UNWRAP	No	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_SIGN	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY_RECOV	No	No		

C_UnwrapKey				
Attributes	Key Type		Default Value	Note
	Generic Secret	AES		
CKA_DERIVE	Yes	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_START_DATE	Yes	Yes		
CKA_END_DATE	Yes	Yes		
CKA_MODULUS	No	No		
CKA_MODULUS_BIT	No	No		
CKA_PUBLIC_EXPON	No	No		
CKA_PUBLIC_KEY_IN	No	No		
CKA_VALUE_LEN	Yes	Yes		Mandatory template attribute.
CKA_EXTRACTABLE	Yes	Yes	FALSE	NVIDIA limitation. Default False on Unwrap.
CKA_LOCAL	Read-only	Read-only	FALSE	Must not be template attribute.
CKA_NEVER_EXTRACTABLE	Read-only	Read-only	FALSE	Must not be template attribute.
CKA_ALWAYS_SENSITIVE	Read-only	Read-only	FALSE	Must not be template attribute.
CKA_KEY_GEN_MEC	Read-only	Read-only	CK_UNAVAILABLE_INFORMATION	Must not be template attribute.
CKA_MODIFIABLE	Yes	Yes	TRUE	
CKA_COPYABLE	Yes	Yes	TRUE	
CKA_DESTROYABLE	Yes	Yes	TRUE	

C_UnwrapKey				
Attributes	Key Type		Default Value	Note
	Generic Secret	AES		
CKA_EC_PARAMS	No	No		
CKA_EC_POINT	No	No		
CKA_WRAP_WITH_TI	Yes	Yes	FALSE	
CKA_WRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.
CKA_UNWRAP_TEMPLATE	No	No		NVIDIA limitation. Not supported.
CKA_ALLOWED_MECHANISM	Yes	Yes		Mandatory template attribute.
CKA_NVIDIA_CALLEE	Read-only	Read-only	FALSE	

Unwrap Private Key Attributes Support with CKM_AES_CBC

The table below lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type being created.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for the specific key type.
No	Indicates that PKCS#11 library does not support the attribute for the specific key type.
Read-only	The attribute is set to read-only for the specific key type.
	An empty cell in Default Value column indicates that there is no specific value assigned to the attribute.
(Result of library function)	Indicates that the PKCS#11 library determines the attribute value.

C_UnwrapKey			
Attributes	Key Type	Default Value	Note
	EC Private		
CKA_CLASS	Yes	CKO_PRIVATE_KEY	Mandatory template attribute.
CKA_TOKEN	Read-only	FALSE	NVIDIA limitation. Only Ephemeral keys can be unwrapped if attribute template is supported.
CKA_PRIVATE	Read-only	TRUE	NVIDIA limitation. All objects are private.
CKA_LABEL	Yes		
CKA_VALUE	No		
CKA_TRUSTED	No		
CKA_CHECK_VALUE	No		
CKA_KEY_TYPE	Yes		Mandatory template attribute.
CKA_SUBJECT	No		NVIDIA limitation. Attribute not supported.
CKA_ID	Yes		Mandatory template attribute.
CKA_SENSITIVE	Read-only	TRUE	NVIDIA limitation. No access to private key material.
CKA_ENCRYPT	No		
CKA_DECRYPT	Read-only	FALSE	NVIDIA limitation. Private key decryption is not supported.
CKA_WRAP	No		
CKA_UNWRAP	Read-only	FALSE	NVIDIA limitation. Private key unwrap is not supported.

C_UnwrapKey			
Attributes	Key Type	Default Value	Note
CKA_SIGN	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_VERIFY	No		
CKA_VERIFY_RECOVER	No		
CKA_DERIVE	Yes	FALSE	NVIDIA limitation. Observe single purpose rules.
CKA_START_DATE	Yes		
CKA_END_DATE	Yes		
CKA_MODULUS	No		
CKA_MODULUS_BITS	No		
CKA_PUBLIC_EXPONENT	No		
CKA_PUBLIC_KEY_INFO	No		
CKA_VALUE_LEN	No		
CKA_EXTRACTABLE	Yes	FALSE	NVIDIA limitation. Default False on Unwrap.
CKA_LOCAL	Read-only	FALSE	Must not be template attribute.
CKA_NEVER_EXTRACTABLE	Read-only	FALSE	Must not be template attribute.
CKA_ALWAYS_SENSITIVE	Read-only	FALSE	Must not be template attribute.
CKA_KEY_GEN_MECHANISM	Read-only	CK_UNAVAILABLE_INFORMATION	Must not be template attribute.
CKA_MODIFIABLE	Yes	TRUE	
CKA_COPYABLE	Yes	TRUE	
CKA_DESTROYABLE	Yes	TRUE	

C_UnwrapKey			
Attributes	Key Type	Default Value	Note
CKA_EC_PARAMS	Yes		Mandatory template attribute.
CKA_EC_POINT	No		
CKA_WRAP_WITH_TRUSTED	Yes	FALSE	
CKA_WRAP_TEMPLATE	No		
CKA_UNWRAP_TEMPLATE	No		NVIDIA limitation. Not supported
CKA_ALLOWED_MECHANISM	Yes		Mandatory template attribute
CKA_NVIDIA_CALLER_NO	No		

Copy Key Attributes Support

The following table lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type being copied.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for the specific key type.
No	Indicates that PKCS#11 library does not support the attribute for the specific key type.
Read-only	The attribute is set to read-only for the specific key type.
	An empty cell in Default Value column indicates that there is no specific value assigned to the attribute.
(Result of library function)	Indicates that the PKCS#11 library determines the attribute value.

C_CopyObject								
Attributes	Key Type						Default Value	Note
	EC Priv	EC Public	RSA Public	Ger Sec	AES			
CKA_CLASS	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied		
CKA_TOKEN	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	NVIDIA limitation. A token key cannot be copied into a session key or vice versa.	
CKA_PRIVATE	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied		
CKA_LABEL	Yes	Yes	Yes	Yes	Yes	Inherited from Object being copied		
CKA_VALUE	No	No	No	Read-only	Read-only	Inherited from Object being copied		
CKA_TRUSTED	No	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied		
CKA_CHECK_VALUE	No	No	No	No	No		NVIDIA limitation. Attribute not supported.	
CKA_KEY_TYPE	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied		
CKA_SUBJECT	No	No	No	No	No		NVIDIA limitation. Attribute not supported.	
CKA_ID	Yes	Yes	Yes	Yes	Yes		Mandatory template attribute.	
CKA_SENSITIVE	Read-only	No	No	Read-only	Read-only	Inherited from Object being copied		
CKA_ENCRYPT	No	Read-only	Read-only	No	Read-only	Inherited from Object being copied	NVIDIA limitation. Key usage immutability.	

C_CopyObject							
Attributes	Key Type				Default Value		Note
	EC Priv	EC Public	RSA Public	Ger Sec	AES		
CKA_DECRYPT	Read-only	No	No	No	Read-only	Inherited from Object being copied	NVIDIA limitation. Key usage immutability.
CKA_WRAP	No	Read-only	Read-only	No	Read-only	Inherited from Object being copied	NVIDIA limitation. Key usage immutability.
CKA_UNWRAP	Read-only	No	No	No	Read-only	Inherited from Object being copied	NVIDIA limitation. Key usage immutability.
CKA_SIGN	Read-only	No	No	Read-only	Read-only	Inherited from Object being copied	NVIDIA limitation. Key usage immutability.
CKA_SIGN_RECOVER	No	No	No	No	No		NVIDIA limitation. Attribute not supported for private keys.
CKA_VERIFY	No	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	
CKA_VERIFY_RECOVER	No	No	No	No	No		NVIDIA limitation. Attribute not supported.
CKA_DERIVE	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	
CKA_START_DATE	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	
CKA_END_DATE	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	
CKA_MODULUS	No	No	Read-only	No	No	Inherited from Object being copied	
CKA_MODULUS_BIT	No	No	Read-only	No	No	Inherited from Object being copied	

C_CopyObject							
Attributes	Key Type					Default Value	Note
	EC Priv	EC Public	RSA Public	Ger Sec	AES		
CKA_PUBLIC_EXPONENT	No	No	Read-only	No	No	Inherited from Object being copied	
CKA_PUBLIC_KEY_INNER	No	No	No	No	No		NVIDIA limitation. Attribute not supported
CKA_VALUE_LEN	No	No	No	Read-only	Read-only	Inherited from Object being copied	
CKA_EXTRACTABLE	Read-only	No	No	Read-only	Read-only	Inherited from Object being copied	
CKA_LOCAL	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	
CKA_NEVER_EXTRACTABLE	Read-only	No	No	Read-only	Read-only	Inherited from Object being copied	
CKA_ALWAYS_SENSITIVE	Read-only	No	No	Read-only	Read-only	Inherited from Object being copied	
CKA_KEY_GEN_MEC	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	
CKA_MODIFIABLE	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	
CKA_COPYABLE	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	
CKA_DESTROYABLE	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied	
CKA_EC_PARAMS	Read-only	Read-only	No	No	No	Inherited from Object being copied	
CKA_EC_POINT	No	Read-only	No	No	No	Inherited from Object being copied	

C_CopyObject								
Attributes	Key Type						Default Value	Note
	EC Priv	EC Public	RSA Public	Ger Sec	AES			
CKA_WRAP_WITH_TEMPLATE	Read-only	No	No	Read-only	Read-only	Inherited from Object being copied		
CKA_WRAP_TEMPLATE	No	No	No	No	No			NVIDIA limitation. Not supported.
CKA_UNWRAP_TEMPLATE	No	No	No	No	No			NVIDIA limitation. Not supported.
CKA_ALLOWED_MECHANISMS	Read-only	Read-only	Read-only	Read-only	Read-only	Inherited from Object being copied		
CKA_ALWAYS_AUTH	No	No	No	No	No			NVIDIA limitation. Not supported.
CKA_NVIDIA_USER_INTERFACE	No	No	No	Read-only	Read-only	Inherited from Object being copied		

Set Attributes Support



Note:

Only a single attribute may be set at a time.

The following table lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type operation.

Table Entry	Meaning
Yes	Indicates that PKCS#11 Library supports set attribute for the specific key type.
No	Indicates that PKCS#11 Library does not support set attribute for the specific key type.

C_SetAttributeValue

Attributes	Key Type			Note
	EC Public	RSA Public	AES	
CKA_LABEL	Yes	Yes	Yes	NVIDIA limitation. Set a single attribute at a time.
CKA_TRUSTED	No	No	No	NVIDIA limitation. Cannot create a trusted wrapping key at runtime.
CKA_CHECK_VALUE	No	No	No	NVIDIA limitation.
CKA SUBJECT	No	No	No	NVIDIA limitation.
CKA_ID	Yes	Yes	Yes	NVIDIA limitation. Set a single attribute at a time.
CKA_SENSITIVE	No	No	No	NVIDIA limitation.
CKA_ENCRYPT	No	No	No	NVIDIA limitation. Observe single purpose immutability rule.
CKA_DECRYPT	No	No	No	NVIDIA limitation. Observe single purpose immutability rule.
CKA_WRAP	No	No	No	NVIDIA limitation. Observe single purpose immutability rule.
CKA_UNWRAP	No	No	No	NVIDIA limitation. Observe single purpose immutability rule.
CKA_SIGN	No	No	No	NVIDIA limitation. Observe single purpose immutability rule.
CKA_SIGN_RECOVER	No	No	No	NVIDIA limitation.
CKA_VERIFY	No	No	No	NVIDIA limitation. Observe single purpose immutability rule.
CKA_VERIFY_RECOVER	No	No	No	NVIDIA limitation.
CKA_DERIVE	No	No	No	NVIDIA limitation. Observe single purpose immutability rule.

C_SetAttributeValue

Attributes	Key Type			Note
	EC Public	RSA Public	AES	
CKA_START_DATE	No	No	No	NVIDIA limitation.
CKA_END_DATE	No	No	No	NVIDIA limitation.
CKA_PUBLIC_KEY_INFO	No	No	No	NVIDIA limitation.
CKA_EXTRACTABLE	No	No	No	NVIDIA limitation.
CKA_NVIDIA_USER_NONCE	No	No	No	

Get Attributes Support

The following table lists attributes that differ by key types. It indicates whether a given attribute in a template is supported for a particular key type.

Table Entry	Meaning
Yes	Indicates that PKCS#11 Library supports the attribute for the specific key type.
No	Indicates that PKCS#11 Library does not support the attribute for the specific key type.
No Get	Indicates that the attribute is sensitive and cannot be revealed.

C_GetAttributeValue

Attributes	Key Type	Note				
		EC Private	EC Public	RSA Public	Generic Secret	AES
CKA_CLASS	Yes	Yes	Yes	Yes	Yes	
CKA_TOKEN	Yes	Yes	Yes	Yes	Yes	
CKA_PRIVATE	Yes	Yes	Yes	Yes	Yes	
CKA_LABEL	Yes	Yes	Yes	Yes	Yes	

C_GetAttributeValue						
Attributes Key Type						Note
	EC Private	EC Public	RSA Public	Generic Secret	AES	
CKA_VALUE	No	No	No	No Get	No Get	NVIDIA limitation. Attribute always sensitive and not returned.
CKA_TRUST	No	Yes	Yes	Yes	Yes	
CKA_CHECK	No	No	No	No	No	NVIDIA limitation. Attribute not supported.
CKA_KEY_TYPE	Yes	Yes	Yes	Yes	Yes	
CKA_SUBJECT	No	No	No	No	No	NVIDIA limitation. Attribute not supported.
CKA_ID	Yes	Yes	Yes	Yes	Yes	
CKA_SENSITIVITY	Yes	No	No	Yes	Yes	
CKA_ENCRYPT	No	Yes	Yes	No	Yes	
CKA_DECRYPT	Yes	No	No	No	Yes	
CKA_WRAP	No	Yes	Yes	No	Yes	
CKA_UNWRAP	Yes	No	No	No	Yes	
CKA_SIGN	Yes	No	No	Yes	Yes	

C_GetAttributeValue						
Attributes Key Type					Note	
	EC Private	EC Public	RSA Public	Generic Secret	AES	
CKA_SIGN_	No	No	No	No	No	NVIDIA limitation. Attribute not supported for Private keys.
CKA_VERIFY	No	Yes	Yes	Yes	Yes	
CKA_VERIFY	No	No	No	No	No	NVIDIA limitation. Attribute not supported for public keys.
CKA_DERIV	Yes	Yes	Yes	Yes	Yes	
CKA_START	Yes	Yes	Yes	Yes	Yes	
CKA_END_	Yes	Yes	Yes	Yes	Yes	
CKA_MODU	No	No	Yes	No	No	
CKA_MODU	No	No	Yes	No	No	
CKA_PUBLI	No	No	Yes	No	No	
CKA_PUBLI	No	No	No	No	No	NVIDIA limitation. Attribute not supported for public keys.
CKA_VALUE	No	No	No	Yes	Yes	
CKA_EXTRA	Yes	No	No	Yes	Yes	
CKA_LOCAL	Yes	Yes	Yes	Yes	Yes	

C_GetAttributeValue					
Attributes Key Type					Note
	EC Private	EC Public	RSA Public	Generic Secret	AES
CKA_NEVER	Yes	No	No	Yes	Yes
CKA_ALWA	Yes	No	No	Yes	Yes
CKA_KEY_G	Yes	Yes	Yes	Yes	Yes Contains a valid value only if CKA_LOCAL is TRUE. Else is CK_UNAVAILABLE_INFORMATION
CKA_MODI	Yes	Yes	Yes	Yes	Yes
CKA_COPY	Yes	Yes	Yes	Yes	Yes
CKA_DESTF	Yes	Yes	Yes	Yes	Yes
CKA_EC_PA	Yes	Yes	No	No	No NVIDIA limitation. Contains CK_UNAVAILABLE_INFORMATION
CKA_EC_PC	No	Yes	No	No	No
CKA_WRAP	Yes	No	No	Yes	Yes
CKA_WRAP	No	No	No	No	No NVIDIA limitation. Not supported.
CKA_UNWF	No	No	No	No	No NVIDIA limitation. Not supported.

C_GetAttributeValue						
Attributes Key Type						Note
		EC Private	EC Public	RSA Public	Generic Secret	AES
CKA_ALLO	Yes	Yes	Yes	Yes	Yes	
CKA_ALWA	No	No	No	No	No	NVIDIA limitation. Not supported.
CKA_NVIDI	No	No	No	Yes	Yes	

Create Data Object Attributes Support

The following table indicates whether a given attribute in a template is supported for a Data Object being created.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for a Data Object.
No	Indicates that PKCS#11 library does not support the attribute for a Data Object.
Read-only	The attribute is set to read-only for a Data Object.
	An empty cell in Default Value column indicates that there is no specific value assigned to the attribute.
(Result of library function)	Indicates that the attribute value is determined by the PKCS#11 library

C_CreateObject			
Attributes	Data Object	Default Value	Note
CKA_CLASS	Yes	CKO_DATA	Mandatory template attribute.
CKA_TOKEN	Yes	FALSE	
CKA_PRIVATE	Read-only	TRUE	NVIDIA limitation. All objects are private.
CKA_LABEL	Yes		

C_CreateObject

Attributes	Data Object	Default Value	Note
CKA_VALUE	Yes		-
CKA_ID	Yes	-	Mandatory template attribute.
CKA_VALUE_LEN	Read-only	(Result of library function)	Must not be template attribute.
CKA_MODIFIABLE	Yes	TRUE	
CKA_COPYABLE	Yes	TRUE	
CKA_DESTROYABLE	Yes	TRUE	
CKA_APPLICATION	Yes		
CKA_OBJECT_ID	Yes		

Copy Data Object Attributes Support

The table below indicates whether a given attribute in a template is supported for a Data Object being copied.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for a Data Object.
No	Indicates that PKCS#11 library does not support the attribute for a Data Object.
Read-only	The attribute is set to read-only for a Data Object.
	An empty cell in Default Value column indicates that there is no specific value assigned to the attribute.
(Result of library function)	Indicates that the PKCS#11 library determines the attribute value.

C_CopyObject

Attributes	Data Object	Default Value	Note
CKA_CLASS	Read-only	Inherited from Object being copied	-

C_CopyObject			
Attributes	Data Object	Default Value	Note
CKA_TOKEN	Read-only	Inherited from Object being copied	
CKA_PRIVATE	Read-only	Inherited from Object being copied	-
CKA_LABEL	Yes	Inherited from Object being copied	
CKA_VALUE	Yes	Inherited from Object being copied	-
CKA_ID	Yes	-	Mandatory template attribute.
CKA_VALUE_LEN	Read-only	Inherited from Object being copied	-
CKA_MODIFIABLE	Read-only	Inherited from Object being copied	
CKA_COPYABLE	Read-only	Inherited from Object being copied	
CKA_DESTROYABLE	Read-only	Inherited from Object being copied	
CKA_APPLICATION	Read-only	Inherited from Object being copied	
CKA_OBJECT_ID	Read-only	Inherited from Object being copied	

Set Data Object Attributes Support

The following table below indicates whether a given attribute in a template is supported for a Data Object set attribute operation after being created.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports set attribute for a Data Object.
No	Indicates that PKCS#11 library does not support set attribute for a Data Object.

C_SetAttributeValue

Attributes	Data Object	Note
CKA_LABEL	Yes	NVIDIA limitation. Set a single attribute at a time.
CKA_VALUE	Yes	NVIDIA limitation. Set a single attribute at a time.
CKA_ID	Yes	NVIDIA limitation. Set a single attribute at a time.
CKA_APPLICATION	No	
CKA_OBJECT_ID	No	-

Get Data Object Attributes Support

The following table indicates whether a given attribute in a template is supported for a Data Object attribute being fetched after creation.

Table Entry	Meaning
Yes	Indicates that PKCS#11 library supports the attribute for a Data Object.
No	Indicates that PKCS#11 library does not support the attribute for a Data Object.

C_GetAttributeValue

Attributes	Data Object	Note
CKA_CLASS	Yes	
CKA_TOKEN	Yes	
CKA_PRIVATE	Yes	
CKA_LABEL	Yes	
CKA_VALUE	Yes	
CKA_ID	Yes	
CKA_VALUE_LEN	Yes	
CKA_MODIFIABLE	Yes	
CKA_COPYABLE	Yes	

C_GetAttributeValue		
Attributes	Data Object	Note
CKA_DESTROYABLE	Yes	
CKA_APPLICATION	Yes	
CKA_OBJECT_ID	Yes	

Key Exclusive Usage Rules

PKCS#11 library limits key usage attributes such that a key is only usable for a single purpose, or for a single class of purposes. The following purposes and purpose combinations are valid:

- > Encryption (CKA_ENCRYPT)
- > Decryption (CKA_DECRYPT)
- > Encryption and decryption (CKA_ENCRYPT | CKA_DECRYPT)
- > Signature generation (CKA_SIGN)
- > Signature verification (CKA_VERIFY)
- > Signature generation and verification (CKA_SIGN | CKA_VERIFY)
- > Key unwrapping (CKA_UNWRAP)
- > Key wrapping (CKA_WRAP)
- > Key unwrapping and wrapping (CKA_UNWRAP | CKA_WRAP)
- > Key derivation (CKA_DERIVE)

Key Usage Immutability

PKCS#11 library does not allow modification of key usage attributes after key creation.

CKA_ID

PKCS#11 library requires that any CKA_ID generated by the client application satisfies the following constraints:

- > A byte array of CK_BYTEs must be padded with space character to 32 bytes
- > No NULL character
- > Must not start with "NV"
- > Unique

Returns CKR_ATTRIBUTE_VALUE_INVALID if any of these conditions are not met.

Attribute Repeated in Template

PKCS#11 library returns CKR_TEMPLATE_INCONSISTENT if a template for an object specifies the same attribute more than once.

Surplus Attributes in Template

PKCS#11 library returns CKR_TEMPLATE_INCONSISTENT if a template for an object specifies attributes surplus to expectation.

Unwrap Template Not Supported

The attribute CKA_UNWRAP_TEMPLATE is not supported.

Wrap Template Not Supported

The attribute CKA_WRAP_TEMPLATE is not supported.

Unique ID Not Supported

The attribute CKA_UNIQUE_ID is not supported.

8.2.6 PKCS#11 – Sample Application

PKCS#11 library includes sample application code for customer reference to demonstrate use of the following:

- C_GetSlotList to find the slot and token you require.
- C_GetTokenInfo to obtain information about a particular token, token status, and the status of a token's secure storage.
- NVIDIA channel extension APIs C_NVIDIA_InitializeChannel, C_NVIDIA_OpenSession and C_NVIDIA_FinalizeChannel to redirect digest operation on to TZ-SE (QNX only) and to redirect sign and verify operations with CKM_SHA256_HMAC on to a SHA engine (QNX only).
- C_UnwrapKey to provision a wrapped key using CKM_NVIDIA_AES_GCM_KEY_UNWRAP.
- Wrap and unwrap a key using CKM_AES_CBC and retrieval of the IV generated during the wrap operation (the same IV is required to successfully unwrap the key).
- CKM_NVIDIA_AES_CBC_KEY_DATA_WRAP mechanism with a CK_NVIDIA_AES_CBC_KEY_DATA_WRAP_PARAMS mechanism parameter to wrap either one secret key, or a pair of secret keys with custom data interleaved between the two.
- Commit a key to secure storage using C_NVIDIA_CommitTokenObjects.
- Encrypt with CKM_AES_GCM and retrieval of the IV generated during the encrypt operation with C_NVIDIA_EncryptGetIV.
- Mechanisms:
 1. CKM_EDDSA
 2. CKM_SP800_108_COUNTER_KDF
 3. CKM_SHA256
 4. CKM_SHA512
 5. CKM_SHA256_HMAC (QNX only)
 6. CKM_NVIDIA_SP800_56C_TWO_STEPS_KDF

7. CKM_AES_GCM
8. CKM_AES_CMAC
9. CKM_AES_CBC
10. CKM_AES_KEY_GEN
11. CKM_NVIDIA_AES_CBC_KEY_DATA_WRAP
12. CKM_NVIDIA_AES_GCM_KEY_UNWRAP

Refer to the following README for instructions to build the sample application, pkcs11_reference_application:

```
samples/nvpkcs11/external/README
```

8.2.7 PKCS#11 – Implementation Details

Slots and Tokens

A **PKCS#11 token** represents a combination of **persistent object storage**, and access to **cryptographic hardware**. In releases prior to 6.0.5.0, the NVIDIA PKCS#11 implementation supported a single token instance- a single persistent storage area (ID 2) and a single set of hardware (CCPLEX). In 6.0.5.0 and future releases, multiple tokens are supported.

Three types of hardware are supported; these need to be represented in different PKCS#11 tokens, as they represent different hardware:

- > **CCPLEX** (the largest set of cryptographic hardware support).
- > **TSEC**(TSEC is a special hardware that supports AES CMAC sign and verify exclusively. The TSEC hardware can only perform CMAC operations with keys in the safety token. To use TSEC, firstly install all required keys in the dynamic token as token objects, commit them, and then reboot. After this, the safety token will be able to use the previously installed keys for AES CMAC operations.)
- > **FSI** (key management only, no cryptographic operations supported).

There is a requirement for **object access control** for CCPLEX. This is to allow different applications to use the same CCPLEX hardware, but with access to different sets of objects. For example, you can have an application that processes sensor data with one set of keys and a webstore application with a different set of keys. Each application must not be able to access the other set of objects but must execute operations on the same set of cryptographic hardware. This is implemented by having multiple PKCS#11 tokens for CCPLEX hardware, each with their own storage areas and access protection GIDs.

There is a requirement to **protect safety applications from changes** to token objects while they are running (UNECE 156a 7.2.2.1.3). There may also be other, non-safety critical applications that need the ability to change token objects at runtime. To allow for this, each configuration of hardware and access control will have a **dynamic** token view and a **safety** token view. The **dynamic** token allows for token objects to be **added, updated, and deleted**, and once added can be used immediately. The **safety** token has a static view of the content of the persistent storage as it was at boot time - objects can be **accessed, but not altered, added, or deleted** in this view.

To alter the objects in the safety view:

- Make the changes in the dynamic view of the same storage ID
- Call C_NVIDIA_CommitTokenObjects() with no PKCS#11 sessions open on any safety token (this is to prevent safety-critical operations stalling as the commit happens)
- Either reboot, or go through an SC7 cycle (calling C_Finalize() before suspending and C_Initialize() after resuming)

Twenty seven tokens are supported, which will have the following model names listed in the CK_TOKEN_INFO structure:

Model name	GID	Custom Ability	Secure storage instance ID	Supported hardware, safety or dynamic	Notes
NVPKCS11_FSI_DYNAMIC_1_MOD	6002	nvtzvault/ pkcs11ks_dynamic_token_1_ability nvtzvault/crypto_ability	1	FSI, dynamic	
NVPKCS11_CCPLEX_SAFETY_2_M	6003	nvtzvault/ pkcs11ks_safety_token_2_ability nvtzvault/crypto_ability nvvse/ Engines:0-2	2	CCPLEX, safety	S s
NVPKCS11_CCPLEX_DYNAMIC_2_	6004	nvtzvault/ pkcs11ks_dynamic_token_2_ability nvtzvault/crypto_ability nvvse/ Engines:0-2	2	CCPLEX, dynamic	F o
NVPKCS11_TSEC_SAFETY_3_MOD	6005	nvtzvault/ pkcs11ks_safety_token_3_ability nvtzvault/crypto_ability nvvse/ Engines:6	3	TSEC, safety	T s
NVPKCS11_TSEC_DYNAMIC_3_M	6006	nvtzvault/ pkcs11ks_dynamic_token_3_ability nvtzvault/crypto_ability nvvse/ Engines:6	3	TSEC, dynamic	N o o A s C o
NVPKCS11_CCPLEX_SAFETY_4_M	6007	nvtzvault/ pkcs11ks_safety_token_4_ability nvtzvault/crypto_ability nvvse/ Engines:0-2	4	CCPLEX, safety	
NVPKCS11_CCPLEX_DYNAMIC_4_	6008	nvtzvault/ pkcs11ks_dynamic_token_4_ability nvtzvault/crypto_ability nvvse/ Engines:0-2	4	CCPLEX, dynamic	

Model name	GID	Custom Ability	Secure storage instance ID	Supported hardware, safety or dynamic
NVPKCS11_CCPLEX_SAFETY_5_M	6009	nvtzvault/ pkcs11ks_safety_token_5_ability nvtzvault/crypto_ability nvvse/ Engines:0-2	5	CCPLEX, safety
NVPKCS11_CCPLEX_DYNAMIC_5_	6010	nvtzvault/ pkcs11ks_dynamic_token_5_ability nvtzvault/crypto_ability nvvse/ Engines:0-2	5	CCPLEX, dynamic
...				
NVPKCS11_CCPLEX_SAFETY_13_M	6025	nvtzvault/ pkcs11ks_safety_token_13_ability nvtzvault/crypto_ability nvvse/Engines:0-2 nvtzvault/ macsec_tos_ability	13	CCPLEX, safety
NVPKCS11_CCPLEX_DYNAMIC_13	6026	nvtzvault/ pkcs11ks_dynamic_token_13_ability nvtzvault/crypto_ability nvvse/Engines:0-2 nvtzvault/ macsec_tos_ability	13	CCPLEX, dynamic
NVPKCS11_CCPLEX_SAFETY_14_M	6027	nvtzvault/ pkcs11ks_safety_token_14_ability nvtzvault/crypto_ability nvvse/ Engines:0-2	14	CCPLEX, safety
NVPKCS11_CCPLEX_DYNAMIC_14	6028	nvtzvault/ pkcs11ks_dynamic_token_14_ability nvtzvault/crypto_ability nvvse/ Engines:0-2	14	CCPLEX, dynamic

The PKCS#11 sample app demonstrates how to use C_GetTokenInfo to identify and open a token that matches that name (NVPKCS#11_CCPLEX_DYNAMIC_2_MODEL_NAME).

Accessing the Token

To use a token, your application must have access to both keys in a persistent storage area and access to cryptographic hardware engines. The permissions for these are handled using GIDs and QNX custom abilities.

There are 27 unique key storage area custom abilities, one for each of the supported tokens.

There is 1 nvtzvault/crypto_ability custom ability required by every PKCS#11 library application.

To use a CCPLEX token, your application must also have access permission to all three GP-SE channels, by adding nnvse/Engines:0-2 custom abilities.

To use a TSEC token the application requires nnvse/Engines:6 custom ability.

For an application using a CCPLEX token to also have access to the TZ-SE (QNX only) cryptographic hardware, you add nnvse/Engines:3-5 custom abilities.

The PKCS#11 sample app README file demonstrates how to run the sample app as non-root with correct custom ability to use NVPKCS11_CCPLEX_DYNAMIC_2_MODEL_NAME.



Note: The existing client application code must be updated to choose a token explicitly and to use the new GIDs and custom abilities.

Choosing the Token

If your application is **safety related**, it should use a **safety token**. This is to prevent interference from any other applications or updates.

If your application needs to **update objects in secure storage**, it will need to use a **dynamic token**. Note that the safety token will not see those changes until after a reboot or an SC7 cycle has taken place.

If your application requires **access to persistently stored objects**, and also requires that those objects are **not accessible to any other applications**, then use a **unique secure storage instance ID**.

The following example shows how secure storage updates can be performed, and also how the isolation between safety and dynamic tokens works.

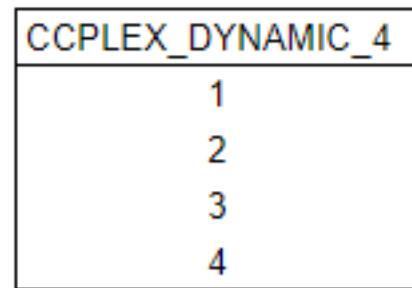
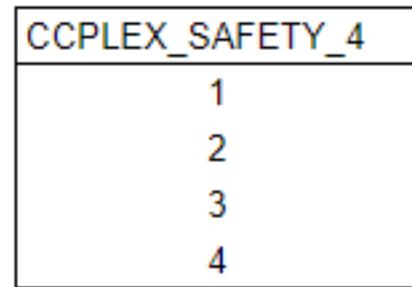
Step 1

System is powered off. Secure storage for ID 4 contains four keys.

Step 2

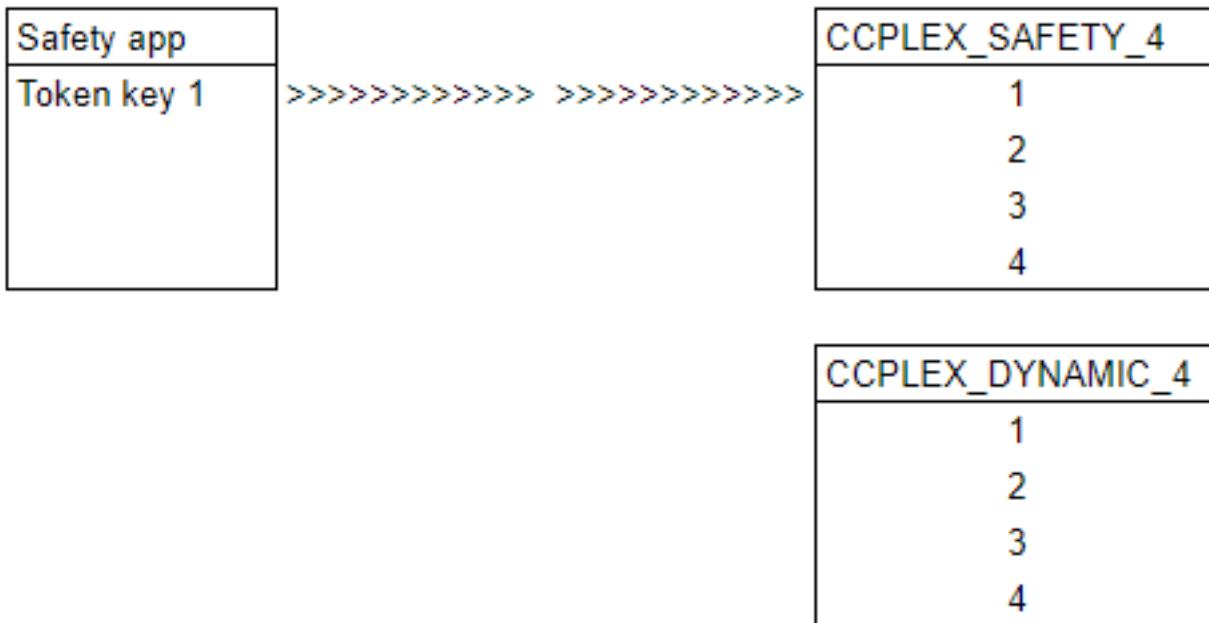
Power on. System boots. Two PKCS11 tokens are created for CCPLEX hardware with secure storage ID 4.

Each token contains a copy of the secure storage content.



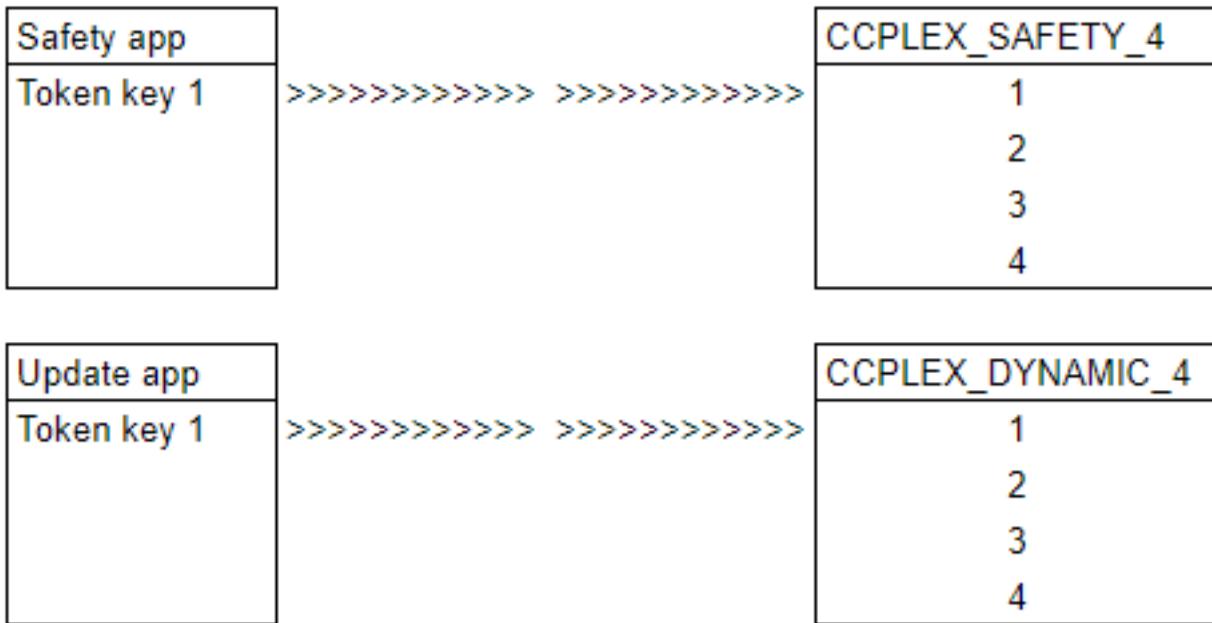
Step 3

A safety critical application connects to the CCPLEX_SAFETY_4 token, and starts to use key 1 for an operation.



Step 4

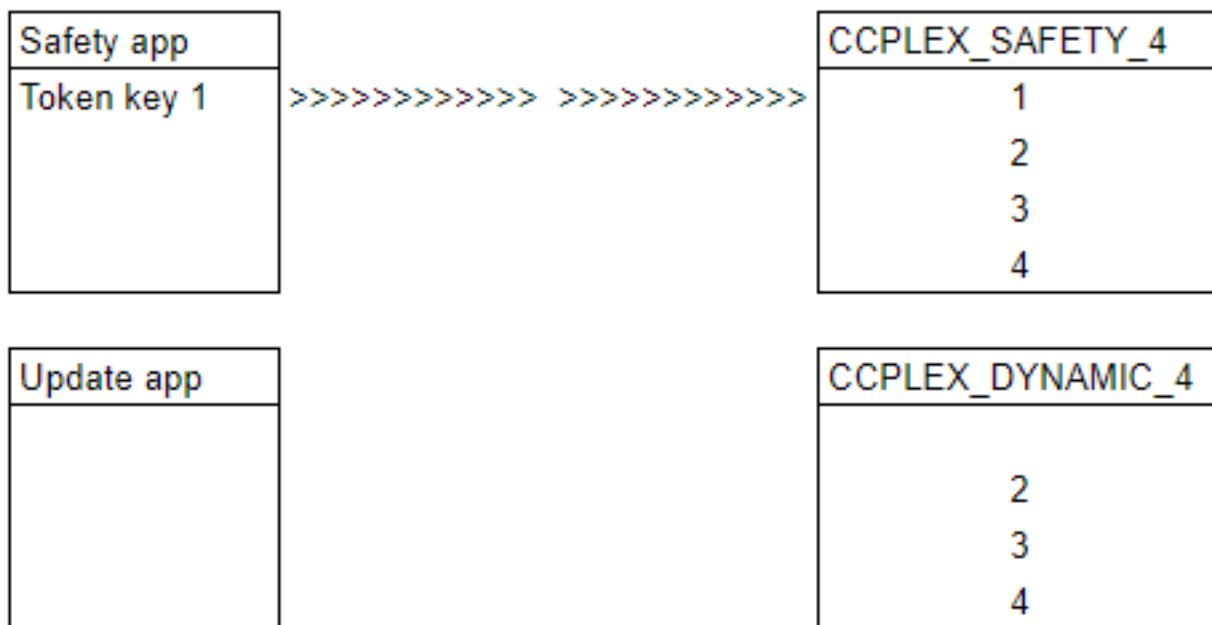
An update app connects to CCPLEX_DYNAMIC_4. This app is not safety critical. It also uses key 1.



Step 5

The update app needs to replace key 1. It will begin by deleting that key. Note that the original key still exists in the safety token, and in secure storage.

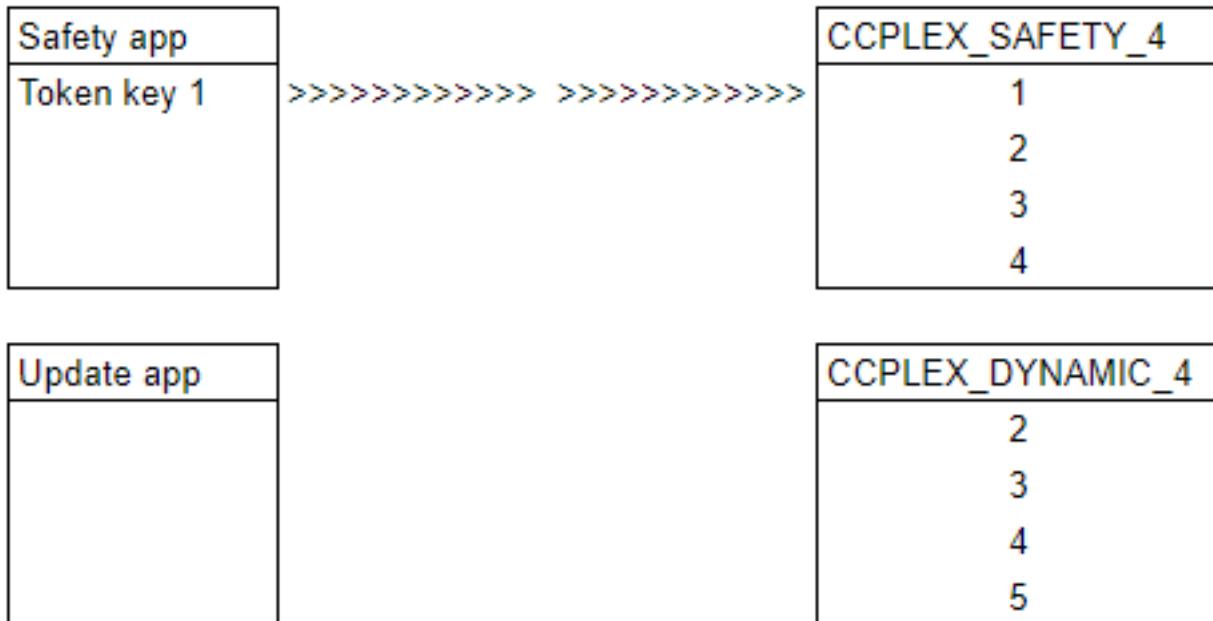
The safety app continues to function without interruption.



Step 6

A new key (5) is added to CCPLEX_DYNAMIC_4. The new key is visible to any app that uses the CCPLEX_DYNAMIC_4 token.

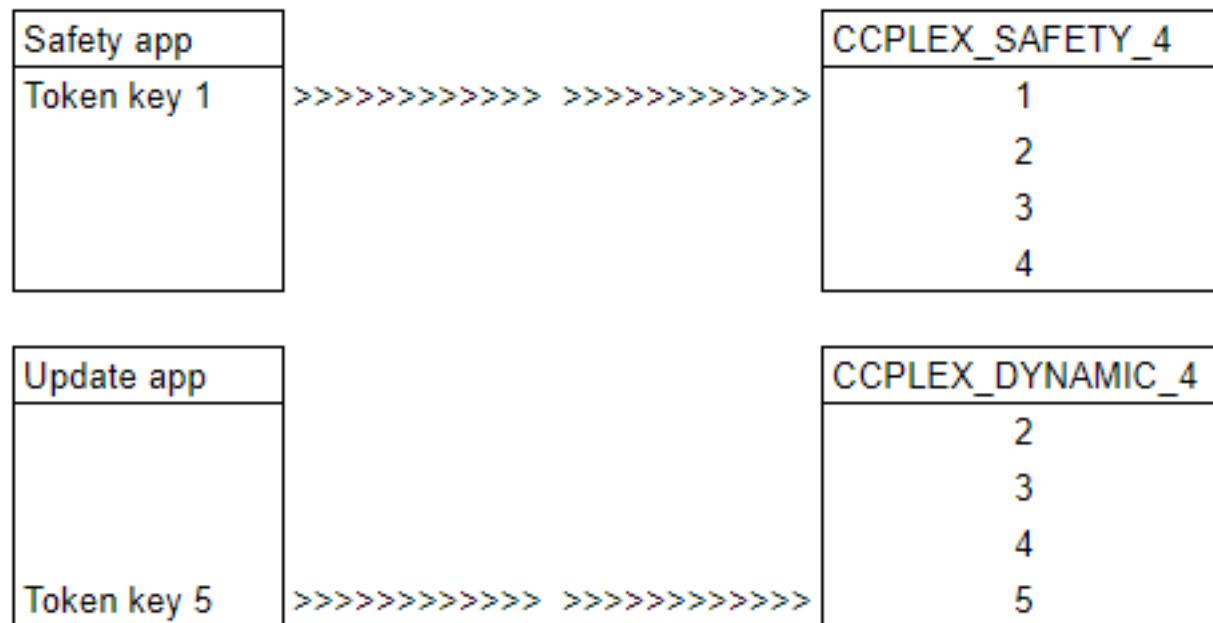
The CCPLEX_SAFETY_4 token has no way to see this key. It is not yet in secure storage, so would be lost if a reboot were to happen.



Step 7

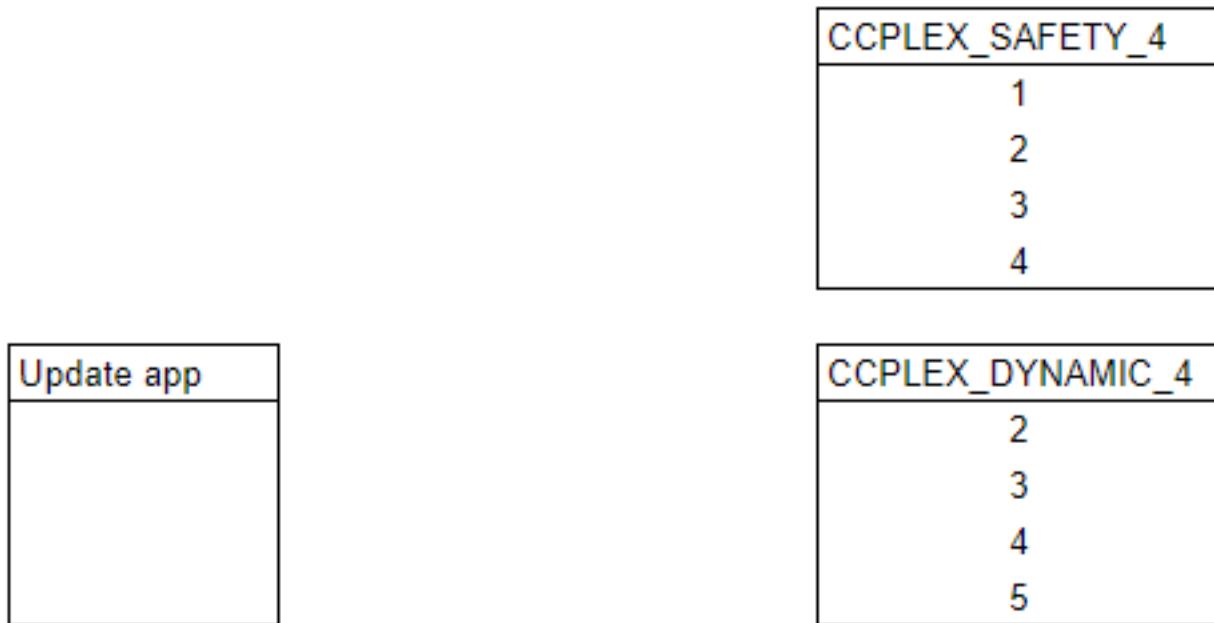
The update app now checks that key 5 is functional by performing some operations with it.

Secure storage cannot be updated yet, as the safety app is still using CCPLEX_SAFETY_4.



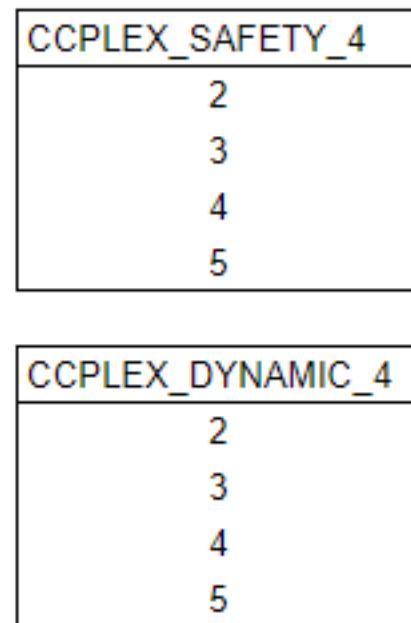
Step 8

Safety app has terminated. The update app then commits the changes to secure storage by calling C_NVIDIA_CommitTokenObjects().



Step 9

A reboot then takes place. Key 1 is no longer usable by CPLEX_SAFETY_4 or CPLEX_DYNAMIC_4, and both tokens can use key 5.



Multi-Threaded Application

PKCS#11 library only supports multi-threaded access by an application using the native operating system primitives.

Session Type

PKCS#11 library supports read/write (R/W) user sessions on dynamic tokens, and read-only (R/O) user sessions on all tokens. To update secure storage in a R/W session, C_NVIDIA_CommitTokenObjects() must be called as outlined previously.

Active Operation Abandonment

PKCS#11 library fails any cryptographic initialization functions for a session if there are any active operations of the same type on that session. The active operation will reset and return to the idle state.

PKCS#11 library additionally allows C_MessageSignInit and C_MessageVerifyInit to be called with pMechanism set to NULL_PTR to terminate an active operation

Operations That Cannot Be Canceled by C_SessionCancel

PKCS#11 library shall return CKR_OPERATION_CANCEL_FAILED if any of the following operations are included in the request to C_SessionCancel:

- > CKF_GENERATE_KEY_PAIR
- > CKF_GENERATE
- > CKF_WRAP
- > CKF_UNWRAP
- > CKF_DERIVE

User Type

PKCS#11 library only supports the user type: normal user.

Read-Only User Authentication

PKCS#11 library supports normal user login to a read only session without the need for a PIN.

pPin from client application is NULL_PTR.

Read/Write User Authentication

PKCS#11 library supports login of an authenticated normal user login to a Read/Write session without the need for a PIN.

pPin from client application is NULL_PTR.

Maximum Input Data Size Limits for AES and Digest Operations

Maximum input data size limits for any operations using GP-SE or TZ-SE (QNX only) cryptographic hardware are determined by the max_buf_size and gcmdec_buf_size configuration parameters described in the "VSE IVC Queue Configuration" topic in the *NVIDIA DRIVE OS QNX SDK Developer Guide*.

Encrypt Input Data Length

PKCS#11 library restricts the data length of all multi-part encrypt update operations using CKM_AES_CBC_PAD mechanism to be a multiple of the cryptographic operation block size. The exception is the last encrypt update part, which does not have to be a multiple of the block size.

Encrypt operations using CKM_AES_CTR mechanism restricts data length to multiples of 16 bytes only.

Decrypt Input Data length

PKCS#11 library restricts the data length of multi-part decrypt update operations to be a multiple of the cryptographic operation block size.

Decrypt operations using CKM_AES_CTR mechanism restricts data length to multiples of 16 bytes only.

Session Object Limit

PKCS#11 library restricts the number of session objects that may be created, generated, derived, or unwrapped across all sessions of a single application to a total of 88 keys split as 64 secret, 8 RSA public, 8 EC public and 8 EC private keys and 8 data objects, subject to the support of those objects by the tokens in use.

Cryptoki Function Calls

PKCS#11 library does not support every function in the Cryptoki API. It has a stub for every unsupported function and returns the value CKR_FUNCTION_NOT_SUPPORTED.

Callback Function Not Supported

PKCS#11 library does not support surrender callbacks.

Find Objects

PKCS#11 library supports finding token and session objects - either all objects, or with a template to narrow the search. The template can have up to one entry each for CKA_TOKEN, CKA_CLASS and CKA_ID, but must have at least one attribute specified (and none repeated).

If a search operation is active, and objects are created or destroyed which fit the scope of the search, the search operation will fail upon a C_FindObjects() call continuing the search and return CKR_NVIDIA_OBJECTS_CHANGED. In this event, there may be duplicate object handles returned. You should re-initialize the search.

Deriving Additional Key

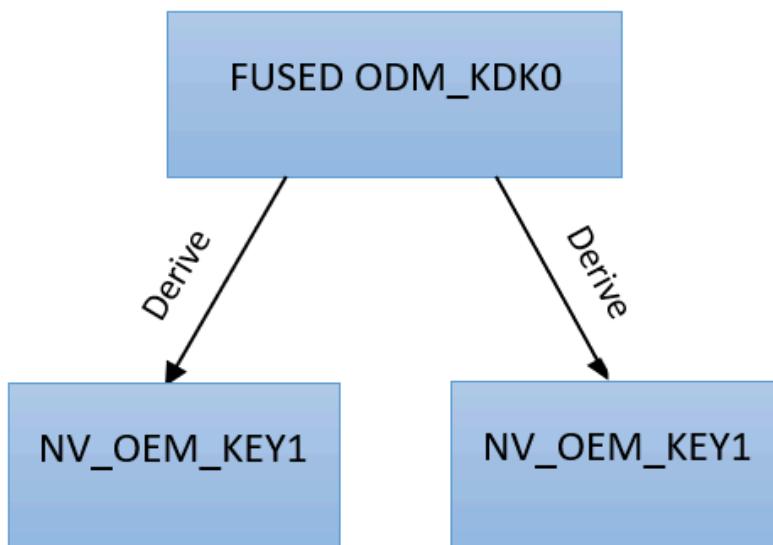
PKCS#11 library limits key derivation to a single derived key per one C_DeriveKey() call.

Derivation From Fuse-based Keys

The support for NV_OEM_KEY1 and NV_OEM_KEY2 for key derivation using SP800-108-Counter-KDF-HMAC-SHA256(Key=FUSE_KEYS_KDK0_0_0 (ODM_KDK0), L=256, Label=<as_required>, Context=<as_required>) scheme has been added. Refer to the following table for derivation data values. The purpose of these keys are set to HMAC in the key manifest, meaning the API is responsible for preparing the input in the format mentioned in the scheme. PKCS#11 Library provides interfaces for key derivation using these two keys.

The two (2) keys (NV_OEM_KEY1 and NV_OEM_KEY2) are derived from ODM_KDK0 using the same SP800-108-Counter-KDF-HMAC-SHA256 scheme. However, the two keys are different in the sense that the label and context used to derive these keys from ODM_KDK0 are different. While NV_OEM_KEY1 is same for all devices, if ODM KDK0 is a class key i.e. same for all devices, NV_OEM_KEY2 is a device-specific key as Exclusive Chip ID (ECID), which is unique to each device, is used as the context for the derivation of this key.

Fused Key Hierarchy



Refer below for derivation data values of NV_OEM_KEY1 and NV_OEM_KEY2

PRF Input	Data Field Identifier	Format	Value
$[i]$	CK_SP800_108_ITERATIO	Big Endian 32 bit integer	Value for NV_OEM_KEY1 & NV_OEM_KEY2 is 0x00000001
<i>Label</i>	CK_SP800_108_BYTE_AR	1 to 32 bytes	NV_OEM_KEY1 - "NV_OEM_DERIVED_1" NV_OEM_KEY2 - "NV_OEM_DERIVED_2"
0x00	CK_SP800_108_BYTE_AR	1 byte	"00" in HexString Format
<i>Context</i>	CK_SP800_108_BYTE_AR	1 to 32 bytes	Value for NV_OEM_KEY1 - "00" in HexString Format and size is 1 Byte Value for NV_OEM_KEY2 - <ECID in HexString Format> and size is 16 Bytes
$[L]$	CK_SP800_108_DKM_LEN	Big Endian 32 bit integer	Key Length for NV_OEM_KEY1 and NV_OEM_KEY2 will be 256 bits, value is 0x00000100

PKCS#11 library supports derivation from the following named fuse-based keys:

CKA_ID		
"NV_OEM_KEY1	"	A general purpose OEM-defined key
"NV_OEM_KEY2	"	A general purpose device-unique key

CKM_SP800_108_COUNTER_KDF Input Parameters

PKCS#11 library limits the counter mode key derivation function, denoted CKM_SP800_108_COUNTER_KDF, to use the following PRF input data definitions.

SP800-108 section 5.1 outlines a sample Counter Mode KDF, which defines the following PRF input:

PRF ($KI, [i] // Label // 0x00 // Context // [L]$)

where $\|$ is the concatenation operation in which the order of the values are defined and KI is the base key being derived from.

The following table lists the PRF input data field types, meanings, limitations, and order that are supported within the CK_PRF_DATA_PARAM structure:

PRF Input	Data Field Identifier	Format	Description	Comment
$[i]$	CK_SP800_108_ITERATION_VARIABLE	Big Endian 8 or 32 bits wide	Counter integer that is the iteration variable. Value shall be 1 for SHA_256_HMAC or AES_CMAC PRF	Default: Use 32 bits wide. 8 bits wide support is for MACSEC wpa-supplicant application
<i>Label</i>	CK_SP800_108_BYTE_ARRAY	1 to 32 bytes	Client Application supplied byte array that identifies the purpose for the derived keying material. The byte 0x00 is not allowed.	
0x00	CK_SP800_108_BYTE_ARRAY	1 byte	An all zero octet. Used to indicate a separation of different variable length data fields	
<i>Context</i>	CK_SP800_108_BYTE_ARRAY	1 to 32 bytes	Client Application supplied byte array containing the information related to the derived key. The byte 0x00 is not allowed.	
$[L]$	CK_SP800_108_DKM_LENGTH	Big Endian 16 or 32 bits wide	An integer specifying the length (in bits) of the derived key.	Default: Use 32 bits wide. 16 bits wide support is for MACSEC wpa-supplicant application

Secret Key Material Protection

PKCS#11 library does not allow access to secret key material or secret key check value.

Added Allowed Function Return Values

PKCS#11 library allows CKR_OPERATION_ACTIVE return value for C_Digest, C_Encrypt, and C_Decrypt functions. The current operation will reset and return to idle state

upon returning CKR_OPERATION_ACTIVE. The approach taken warns of a potential programming error rather than silently accepting it.

PKCS#11 library allows CKR_MECHANISM_INVALID return value for C_xxUpdate, C_xxFinal if C_xxInit is called with mechanism CKM_AES_GCM.

PKCS#11 library allows CKR_OPERATION_NOT_INITIALIZED return value for C_xxMessageBegin, C_EncryptMessage functions.

PKCS#11 library allows CKR_OPERATION_ACTIVE return value for C_xxMessageNext, C_EncryptMessage, C_DecryptMessage functions. The current operation will reset and return to idle state upon returning CKR_OPERATION_ACTIVE. The approach taken warns of a potential programming error rather than silently accepting it.

PKCS#11 library allows CKR_SESSION_READ_ONLY_EXISTS return value for C_OpenSession function.

PKCS#11 library allows CKR_ARGUMENTS_BAD return value for C_EncryptMessageBegin function.

PKCS#11 library allows CKR_MECHANISM_INVALID return value for C_xxMessageBegin functions.

PKCS#11 library allows CKR_FUNCTION_NOT_SUPPORTED return value for C_SignUpdate, C_SignFinal, C_VerifyUpdate and C_VerifyFinal functions.

PKCS#11 library allows C_VerifyMessageNext to continue on CKR_SIGNATURE_INVALID.

Symmetric Key Block Mode

When using PKCS#11 APIs to generate, derive, or unwrap symmetric keys, users of DRIVE OS must specify at most one block mode in the CKA_ALLOWED_MECHANISMS template attribute for any particular key.

Exception: Specifying CKM_AES_CBC and CKM_AES_CBC_PAD for the same key is acceptable.

CMAC Message Length

PKCS#11 library does not support Signature requests using CKM_AES_CMAC with a 0 (zero)-length message.

Mutually Exclusive KDF Mechanism Support

When using PKCS#11 APIs to derive from a base key, that base key can support either one of CKM_SP800_108_COUNTER_KDF or CKM_NVIDIA_SP800_56c_TWO_STEPS_KDF but not both together.

C_Finalize Prior to SC7 Entry

All applications that use PKCS#11 library must call C_Finalize prior to SC7 entry. If this is not done, then you will receive unexpected errors when attempting to use PKCS#11 APIs.

C_WrapKey with CKM_AES_CBC output IV handling

For C_WrapKey with CKM_AES_CBC mechanism the IV buffer provided by the caller as part of the mechanism parameters is an output and is overwritten by an internally generated random IV.

GCM with zero-length Data

PKCS#11 library supports encrypt or decrypt operations using CKM_AES_GCM with 0 (zero)-length data.

8.3 Security Engine

The Security Engine is a general-purpose hardware accelerator for performing Cryptographic Operations and it is virtualized through SE Virtualization Software. SE Virtualization software architecture is based on para-virtualization technique, where a dedicated virtualization system partition (a.k.a. SE resource manager server OR SE Server) is responsible for managing the security engine hardware and a message based interface is exposed to the virtual machines for requesting security engine services.

Refer to the PKCS#11 Interface section for more details on performing a hardware offload of cryptographic operations using Security Engine.

Error Reporting

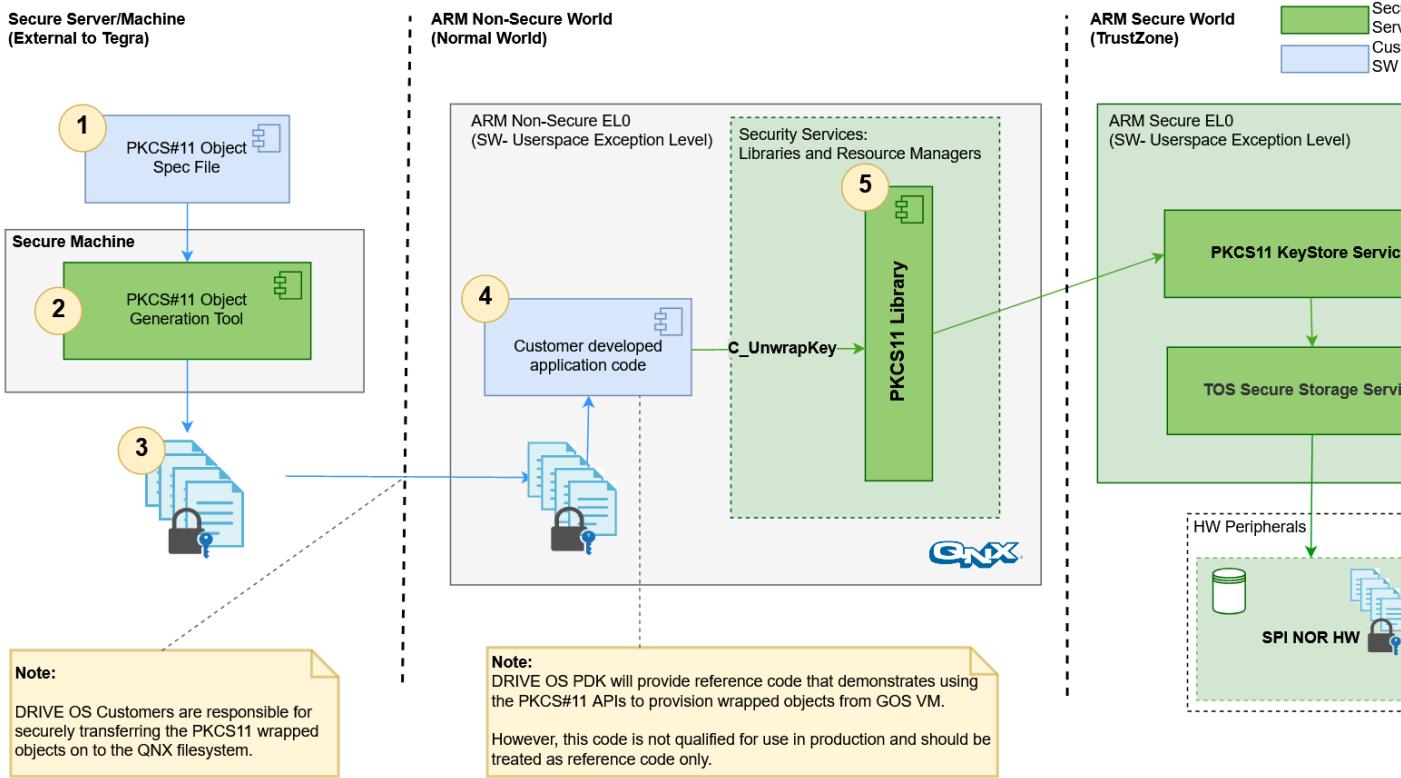
The security engine reports hardware errors to HSM directly. CCPLEX does not handle error reporting and hardware errors.

8.4 Persistent Key Object Support

NVIDIA DRIVE OS Security Services provide the ability to persistently store custom key objects on dedicated secure storage media. The format of the objects adheres to the PKCS#11 specification. The NVIDIA DRIVE OS PDK provides users the ability to generate customized wrapped PKCS#11 Key Objects and reference code that

demonstrates how to provision said objects into the dedicated secure SPI-NOR hardware.

Provisioning of Persistent PKCS#11 Objects



The following section describes the annotations in the “Provisioning of Persistent PKCS#11 Objects” image above.

PKCS#11 Object Spec Files

These files are the input to the PKCS#11 Object Generation Tool that specify secret key material and metadata associated with each key object. These input files must be generated in a secure environment to prevent disclosure of secret key material and associated metadata.

Refer to the [Generating PKCS#11 Key Objects](#) section for more details.

PKCS#11 Object Generation Tool

The PKCS#11 Object Generation Tool takes the inputs from the PKCS#11 Spec files and a secret key (refer to the Derivation From Fuse-based Keys section under [PKCS#11 – Implementation Details](#)) to generate binaries associated with each key object.

Refer to the Generating PKCS#11 Key Objects section for more details.



Important:

The PKCS#11 Object Generation Tool is implemented in a way to facilitate deployment on custom HSM solutions. By default, all cryptographic primitives use openSSL but are abstracted in a separate source file and can be easily replaced with HSM-specific constructs.



Important:

PKCS#11 Key Object format is not guaranteed to be forward compatible and is subject to change in subsequent releases. The PKCS#11 Object Generation Tool ensures that the generated object format is aligned with DRIVE OS internal services.

Please consult with an NVIDIA representative before making changes to the object format. Any modifications to the PKCS#11 Object format in the generation tool may result in a failure during key object provisioning.

Wrapped PKCS#11 Key Objects

These are the binary outputs of the PKCS#11 Object Generation Tool. These wrapped objects are authenticated and encrypted and must be securely transferred over to the target file system where the application will access them.

Application to provision PKCS#11 Key Objects into Secure SPI-NOR hardware

Once the binary output files generated using the PKCS#11 Object Generation Tool are moved over to the Guest OS filesystem, and application must read these binaries and provision them into the Secure SPI-NOR hardware using the following PKCS#11 API:

`C_UnwrapKey()`

Refer to the [Provisioning PKCS#11 Key Objects](#) section for more information.

PKCS#11 Library (QNX)

The user-space library that exposes Security Services to applications in the QNX Guest OS. Refer to the [PKCS#11 Interface](#) for a detailed description and documentation.

Secure SPI-NOR Hardware

The dedicated secure storage media that stores the key objects persistently. This is the destination of the UnwrapKey operation.

Confidentiality, Authenticity and Integrity

The secure storage media, along with its software driver, provides confidentiality, authenticity and integrity for the objects stored using device-specific keys. Attackers cannot read out any part of the objects in plaintext. When there is tampering against the media, errors are returned to the PKCS#11 library.

Rollback Detection

The secure storage media, along with its software driver, provides rollback attack detection for the stored PKCS#11 Key Objects. When there are rollback attacks against the media, errors are returned to the PKCS#11 library.

SPI Error Reporting

When uncorrectable errors happen during Secure SPI-NOR data transfer through SPI bus, these errors are reported to FSI.

Configurable Persistent Storage Capacity

The SPI-NOR space is divided into multiple secure storage instances on top of which the tokens are developed. Access (Read/Write) to each such instance is provided to client through a pair of Safety and Dynamic token (refer to: [PKCS#11 – Implementation Details](#)). The DRIVE OS supports configuring the maximum number of different PKCS#11 Persistent Objects that can be stored in an instance through PCT. The PCT is expected to be configured with the maximum number of objects for each supported PKCS#11 object type.

Update the `tos_keystore_conf` structure of the `drive_av` PCT guest configuration to configure secure storage instances with the required Object Capacity. The `tos_keystore_conf` is defined in the `pct` and details of the structure are below. In the PCT and some other scripts, we use the term `group` to refer to a secure storage instance for convenience, so the two terms can be treated interchangeably:

- > `num_groups`
 - Description:
 - Specified the num of groups that would be configured through PCT
 - Values:
 - Can take from the range 0 to 6 (i.e., a maximum of six groups can be configured through PCT)
- > `ss_group_config`
 - Object Capacity configuration of a particular Secure Storage Group and contains the below attributes
 - `ss_group_id`
 - Description
 - > ID of the Secure Storage Group that needs to be configured
 - Values
 - > Can take from the range 1 to 6
 - `sym_key_capacity`
 - Description
 - > Max number of Symmetric keys the Secure Storage Group needs to support
 - Values
 - > Valid values are determined as below (Validity of a PCT configuration)

- rsa_pub_key_capacity
 - Description
 - > Max number of RSA Public Keys the Secure Storage Group needs to support
 - Values
 - > Valid values are determined as below (Validity of a PCT configuration)
- ecc_pub_key_capacity
 - Description
 - > Max number of ECC Public keys the Secure Storage Group needs to support
 - Values
 - > Valid values are determined as below (Validity of a PCT configuration)
- ecc_priv_key_capacity
 - Description
 - > Max number of ECC Private Keys the Secure Storage Group needs to support
 - Values
 - > Valid values are determined as below (Validity of a PCT configuration)
- generic_data_object_capacity
 - Description
 - > Max number of Generic Data Objects the Secure Storage Group needs to support
 - Values
 - > Valid values are determined as below (Validity of a PCT configuration)
- > ss_group_cfg[PCT_MAX_SS_GROUP_COUNT = 6]
 - Description:
 - An array of ss_group_config to configure multiple Secure Storage Groups

Examples

Following are examples of Object Capacity and the views of the tos_keystore_conf for the corresponding Object Capacity:

Example 1

If it is required that the...

1. NVPKCS11_TSEC_DYNAMIC_3_MODEL_NAME is expected to support a maximum number of 200 Symmetric Key Objects
2. NVPKCS11_CCPLEX_DYNAMIC_2_MODEL_NAME is expected to support a maximum of 16 Symmetric Keys, 4 RSA Public, 2 ECC Public, 2 ECC Private, 2 Generic Data Objects
3. NVPKCS11_CCPLEX_DYNAMIC_4_MODEL_NAME is expected to support a maximum of 0 Symmetric Keys, 0 RSA Public, 0 ECC Public, 0 ECC Private, 10 Generic Data Objects

The PCT configuration should be:

```
.tos_keystore_conf = {
    .num_groups = 3,
    .ss_group_cfg[0] =
    {
        3U, // Group ID of the Secure Storage Group (for Token
NVPKCS11_TSEC_DYNAMIC_3_MODEL_NAME)
        200U, // Max number of Symmetric keys to be supported on this group
        0U, // Max number of RSA Public Keys to be supported on this group
        0U, // Max number of ECC Public Keys to be supported on this group
        0U, // Max number of ECC Private Keys to be supported on this group
        0U, // Max number of Generic Data Objects to be supported on this group
    },
    .ss_group_cfg[1] =
    {
        2U, // Group ID of the Secure Storage Group (for Token
NVPKCS11_CCPLEX_DYNAMIC_2_MODEL_NAME)
        16U, // Max number of Symmetric keys to be supported on this group
        4U, // Max number of RSA Public Keys to be supported on this group
        2U, // Max number of ECC Public Keys to be supported on this group
        2U, // Max number of ECC Private Keys to be supported on this group
        2U, // Max number of Generic Data Objects to be supported on this group
    },
    .ss_group_cfg[2] =
    {
        4U, // Group ID of the Secure Storage Group (for Token
NVPKCS11_CCPLEX_DYNAMIC_4_MODEL_NAME)
        0U, // Max number of Symmetric keys to be supported on this group
        0U, // Max number of RSA Public Keys to be supported on this group
        0U, // Max number of ECC Public Keys to be supported on this group
        0U, // Max number of ECC Private Keys to be supported on this group
        10U, // Max number of Generic Data Objects to be supported on this group
    },
}
```

Example 2

If it is required that the...

1. NVPKCS11_TSEC_DYNAMIC_3_MODEL_NAME is expected to have a maximum number of 400 Symmetric Key Objects
2. NVPKCS11_CCPLEX_DYNAMIC_2_MODEL_NAME is expected to have 68 Symmetric Keys, 0 RSA Public, 0 ECC Public, 0 ECC Private, 5 Generic Data Objects
3. NVPKCS11_CCPLEX_DYNAMIC_4_MODEL_NAME is expected to have 16 Symmetric Keys, 0 RSA Public, 10 ECC Public, 10 ECC Private, 10 Generic Data Objects

```
.tos_keystore_conf = {
    .num_groups = 3,
    .ss_group_cfg[0] = {2U, 68U, 0U, 0U, 0U, 5U},
    .ss_group_cfg[1] = {3U, 400U, 0U, 0U, 0U, 0U},
    .ss_group_cfg[2] = {4U, 16U, 0U, 10U, 10U, 10U}
}
```

Example 3

If it is expected to support only NVPKCS11_TSEC_DYNAMIC_3_MODEL_NAME and expected to have a maximum number of 300 Symmetric Key Objects

```
.tos_keystore_conf = {
    .num_groups = 4,
    .ss_group_cfg[0] = {1U, 0U, 0U, 0U, 0U, 0U},
    .ss_group_cfg[1] = {2U, 0U, 0U, 0U, 0U, 0U},
    .ss_group_cfg[2] = {3U, 300U, 0U, 0U, 0U, 0U},
    .ss_group_cfg[3] = {4U, 0U, 0U, 0U, 0U, 0U}
}
```

Default Config



Note: If not configured through PCT to a certain Object Capacity, a Secure Storage Group is initialized to a default config, as shown below.

```
.ss_group_cfg[0] = {1U, 500U, 1U, 4U, 4U, 5U}, // Group corresponding to FSI token
.ss_group_cfg[1] = {2U, 68U, 16U, 16U, 16U, 16U}, // Group corresponding to Default
CCPLEX
.ss_group_cfg[2] = {3U, 3200U, 0U, 0U, 0U, 0U}, // Group corresponding to TSEC tokens
.ss_group_cfg[3] = {4U, 16U, 2U, 2U, 2U, 2U},
.ss_group_cfg[4] = {5U, 16U, 2U, 2U, 2U, 2U},
.ss_group_cfg[5] = {6U, 16U, 2U, 2U, 2U, 2U},
```

If required to not allocate any storage to a particular Secure Storage Group, please initialize the particular Secure Storage Group Object Capacity to 0U using the PCT.

Validity of a PCT Configuration

The SPI-NOR available to the DRIVE-OS customers is 432 blocks (each block is 4KB). Dividing this space into multiple groups through the PCT configuration is allowed with certain restrictions. The DRIVE-OS clients are expected to configure the PCT with a valid configuration, where a valid configuration is

- > Usage space of cumulative of all Secure Storage Groups cannot exceed 432 4KB-blocks.
- > Total number of objects (sum of maximum supported of each object type) of a particular secure storage group cannot exceed 4000

The DRIVE-OS customers can confirm the validity of a PCT configuration with the Python3 script `token_size_validation.py` provided. The script `token_size_validation.py` can be used as below to validate a PCT configuration. The script can be found at this path on the DRIVE OS PDK:

```
drive-foundation/tools/security/pkcs11/token_size_validation/token_size_validation.py
```

The PCT configuration validation tool does the following things:

- > Check the configuration does not violate the above restriction
- > Provide the usage space of each token. Given the numbers and types of objects of a token, the script performs a depth-first-search algorithm to find the maximum space the token can occupy in case of worst fragmentation in the filesystem.

- > Provide the remaining available space out of <customer_size> or report the configuration is too big and invalid

The steps to use the tool:

```
python3 token_size_validation.py --current-config <current_config.json> --new-config <new_config.json>
python3 token_size_validation.py --ignore-current-config --new-config <new_config.json>
```

Using a new configuration without comparing it with the existing configuration (via `--ignore-current-config`) should only be used in the following scenarios to avoid existing objects on the persistent SNOR exceeding the limit of the new configuration:

1. The board is a dev board (FUSE_SECURITY_MODE_0 is 0) and the developer is accepting the risk of Drive OS persistent object functionality becoming unfunctional.
2. The board is a dev board (FUSE_SECURITY_MODE_0 is 0) and the developer will use the 'wipe_secure_nor' MB2 BCT flag (adding in 6.0.7) to wipe the Secure SPI-NOR during the first boot after flash.
3. The board is new and hasn't been flashed before, which implies the SNOR is brand new.

Compatibility Restrictions

In production, customer cannot OTA a new version of DRIVE-OS with a PCT configuration that reduces the maximum number of objects supported for a particular group. For example, if in current release Secure Storage Instance 4 allows maximum 20 symmetric key objects, reducing it to 10 in the next OTA release will cause DRIVE OS to treat the existing objects as an error as they exceed the new limitation.

This will result in a functionality failure and will lead to the unavailability of PKCS#11 Persistent Object functionality.

It is strongly recommended to only configure the minimal required numbers of objects in each group to preserve space for future OTA releases.

8.4.1 Secure SPI-NOR Provisioning

The SPI-NOR flash is an external secure NOR flash used by the Trusted Execution Environment on the Tegra device for persistent storage of cryptographic assets. The SPI-NOR flash supports authenticated memory access, which relies on a shared symmetric secret known by both the Trusted Execution Environment and the SPI-NOR flash. FSKP programs this shared secret and device security settings into the flash.

One time SPI-NOR provisioning occurs automatically during the next boot after the board FUSE_SECURITY_MODE is burnt. This is the recommended secure NOR provisioning flow for production boards.

For customer development, a MB2 BCT flag snor_provisioning_dev_only is introduced so that a customer engineer can provision the secure NOR without burning the FUSE_SECURITY_MODE, which is not desired for specific development requirements.

A one-time SPI-NOR provisioning occurs during the next boot after the MB2 BCT `snor_provisioning_dev_only` flag is set to 1.

The MB2 BCT flag can be found at the following location:

```
drive-foundation/platform-config/hardware/nvidia/platform/<board>/common/bct/misc/
tegra234-mb2-bct-auto-common.dtsi
```

After the Secure SPI-NOR is provisioned by `snor_provisioning_dev_only`, **do not** burn the fuse keys (e.g., KDK0, KDK0_TAG), other fuses (FUSE_BOOT_SECURITY_INFO), or FUSE_SECURITY_MODE. Fusing these fuses may cause permanent failure of Secure SPI-NOR. For more information, refer to the Manufacture Programmable Fuses chapter in the *NVIDIA DRIVE OS 6.0 PDK Developer Guide*.

Caveats

1. NOR Provisioning flow always locks down the NOR first before provisioning its keys for security reasons, so the NOR provisioning can only happen once. If a NOR is provisioned, all future triggers to provision the secure NOR are ignored.
2. Do not modify the fuse keys after provisioning the NOR. The shared secret between the NOR and the host, as well as the data stored on the NOR, are encrypted with the fuse keys. Any change to the fuse keys will cause the persistent key object support to fail completely or partially.
3. The recommended flow to provision the secure SPI-NOR is:
 - a. **For developers:** Trigger secure NOR provisioning using MB2 BCT flag `snor_provisioning_dev_only` → Start to use NVIDIA DRIVE OS® persistent key object functionalities (do not burn fuse keys, other fuses, or FUSE_SECURITY_MODE).
 - b. **For production:** Run SPI-NOR Mods test → Burn fuse keys and other fuses → Burn FUSE_SECURITY_MODE, which automatically triggers NOR provisioning during first boot → Start to use NVIDIA DRIVE OS persistent key object functionalities. Do not burn fuse keys, other fuses, or FUSE_SECURITY_MODE.

8.4.2 Disable Provisioning for NOR Less Configurations

On NOR less configurations, persistent key object is not supported because the Secure SPI-NOR chip is not available on the board. Therefore, SPI-NOR provisioning must be disabled by setting the `disable_snor_provisioning` MB2 BCT flag to 1.

The MB2 BCT flag is at the following location:

```
drive-foundation/platform-config/hardware/nvidia/platform/<board>/common/bct/misc/
tegra234-mb2-bct-auto-common.dtsi
```

8.5 Generating PKCS#11 Key Objects

The PKCS#11 Object Generation Tool is a python3-based tool that helps users of DRIVE OS generate wrapped key objects on a host machine to later provision on target platforms.

The tool can be found at this path on the DRIVE OS PDK:

```
drive-foundation/tools/security/pkcs11/keywrap/nv_wrap_keys.py
```

8.5.1 Provisioning PKCS#11 Key Objects

PKCS#11 library also provides sample application code for reference. This reference code demonstrates usage of the C_UnwrapKey API. The reference code shows how to consume the output of the PKCS#11 Object Generation Tool and invoke the C_UnwrapKey API to provision the PKCS#11 Key Objects onto the dedicated storage media.



Important:

DRIVE OS PDK reference code demonstrates using the PKCS#11 APIs to provision wrapped 128-bit AES key objects from Guest OS. However, this code is not qualified for use in production and should be treated as reference code only.

Refer to the following README for instructions to build the sample application, pkcs11_reference_application:

```
samples/nvpkcs11/external/README
```

8.6 Enabling JTAG Support on Secure Targets

On secure targets, where the ODM production fuse has been blown, MB1 locks the JTAG connection interface. To enable the JTAG interface on such a target, a special BCT is required where certain parameters are set.

To enable the JTAG interface on a secure target

1. Set the desired debugging features through the BCT section of the flashing configuration file.
2. Generate a new BCT image.

To enable debugging features through the BCT configuration file

1. In the BCT partition of the flashing configuration file, set the uid attribute to the UID of the target device if needing to update the debug features in u32_secure_debug_control_ecid_checked.

2. Set either `u32_secure_debug_control_not_ecid_checked` or `u32_secure_debug_control_ecid_checked` fields to specify the debugging features desired.

The `u32_secure_debug_control_not_ecid_checked` attribute collection of bit fields are as follows.

Bits	Feature enabled by a '1' bit
31:18	Reserved
17	FSI non-secure NIDEN
16	Reserved
15	APE secure debug
14	DCE secure debug
13	Reserved
12	PVA0 secure debug
11	RCE secure debug
10	SCE secure debug
9	SPE secure debug
8:5	Reserved
4	NIDEN (disabled when DEBUG_AUTHENTICATION[1] fuse set)
3:0	Reserved

The `u32_secure_debug_control_ecid_checked` attribute collection of bit fields are as follows.

Bits	Feature enabled by a '1' bit
31	Ramdump
30:26	Reserved
25	FSI secure NIDEN
24	FSI secure DBGEN
23:20	FSI Cluster core debug [3:0]

Bits	Feature enabled by a '1' bit
19	FSI non-secure HNIDEN
18	FSI non-secure HIDEN
17	FSI non-secure NIDEN
16	FSI non-secure DBGEN
5	DBGEN
4	NIDEN
3	SPIDEN
2	SPNIDEN
1	DEVICEEN
0	JTAG_ENABLE

On secure targets MB1 compares the UID of the chip to the UID in the BCT and enables debug features for `u32_secure_debug_control_ecid_checked` if the UIDs match.

The MB1 enables debug features for `u32_secure_debug_control_non_ecid_checked` without verifying the UID.

An example of a flashing configuration file, that specifies the UID value for a specific target device, to enable the JTAG interface with all features in `u32_secure_debug_control_ecid_checked` except Ramdump, is as follows:

```
[partition]
name=bct
allocation_policy=sequential
filesystem_type=basic
uid=0xa18010016419f105000000000a010380
u32_secure_debug_control_ecid_checked=0x03FF003F
size=0x80000
partition_attribute=0
```

8.7 Linux-Based Disk Encryption

Disk encryption ensures that files are always stored on disk in an encrypted form. The files become available to the operating system and applications in readable form while the system is running and unlocked by a trusted user. An unauthorized user inspecting the contents of the disk directly finds garbled random-looking data instead of the actual files.

With user data encryption enabled, the `/home` directory in the file system is encrypted and user data is available when the system is running. The user `/home` partition is mounted on a separate disk partition and block level encryption is enabled for that disk.

The NVIDIA implementation uses the `dm-crypt` kernel module, which is the standard device-mapper interface for encryption functionality provided by the Linux kernel. It is inserted between the disk driver and the file system to transparently encrypt and decrypt the data blocks.

Consult the Linux documentation on `dm-crypt` at:

https://wiki.archlinux.org/index.php/Disk_encryption

The management of `dm-crypt` is performed with the `dmsetup` user-space utility.

8.7.1 dmsetup

The `dmsetup` utility sets up disk encryption based on `dmcrypt` kernel module.

Consult the Linux documentation on `dmsetup` at:

<https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMCrypt>

8.7.2 User Data Encryption

User data encryption is disabled by default on DRIVE OS Linux. Once enabled, all reads and writes to the `/home` directory are encrypted. On boot, during file system setup, the startup scripts mount the user data in the encrypted `/home` directory.

8.7.2.1 To change the size of the user data disk

1. Modify the size parameter for the `gos0-demo-ufs` partition.

The default size is 10 GB.

2. After boot, dump the disk partitions by executing the command:

```
$df -h
```

The new encrypted partition is displayed with the name `/dev/mapper/home-encrypted`.

3. To obtain more information about the encrypted partition, execute the command:

```
$dmsetup status /dev/mapper/home-encrypted
```

4. Execute this command also provides information about the encrypted partition.

```
$dmsetup status
```

A `systemd` service (`nv_cpu_encrypted_user_partition.service`) mounts the encrypted disk at the `/home` directory on every boot. All accesses to the disk are then encrypted. This occurs transparently without user interference.

8.7.3 Steps to Enable Data Encryption

By default, user data encryption is disabled on the NVIDIA DRIVE[®] OS Linux. Follow these steps to enable:

1. Flash the DRIVE OS Linux file system by using the instructions from the SDK [Flashing](#) section and boot the system.
2. After booting the board, ensure that the `/dev/vblkdev50` partition is visible in `cat /proc/partition`.
3. Enable EFS-related systemd service by running the following commands:


```
> sudo su
> systemctl enable
    nv_cpu_encrypt_run_once.service
```
4. Reboot the board.
5. Check the output of the `mount` command.

Note that `/home` should be mounted on the `/dev/mapper/home-encrypted/` partition.

8.7.4 Data Encryption Impact on Boot Times

On the first boot, after flashing, setting up the encrypted partition takes about 30 seconds. These steps are executed by `nv_cpu_encrypt_run_once.service` systemd service. The setup involves:

- > Creating and encrypting VEK (Volume Encryption key) using PKCS#11 app. VEK is stored in `/etc/nvidia/efs/`
- > Decrypting VEK using PKCS#11 app.
- > Setup encrypted partition using `dmsetup` passing VEK and other information.
- > Initializing the encrypted partition with random data
- > Creating ext4 filesystem on encrypted partition
- > Copying the contents from `/home` to encrypted partition
- > Mounting the encrypted partition on `/home` directory

After the first boot, subsequent reboots do NOT have an impact on boot times.

These steps are executed by `nv_cpu_encrypted_user_partition.service` systemd service. Steps involved in subsequent boot for EFS are as follows:

- > Decrypting VEK using PKCS#11 app.
- > Mounting the encrypted partition on `/home` directory

8.7.4.1 Encryption Algorithm

The cipher mode used for disk encryption is `aes-cbc-essiv:sha256`. The cipher generator consists of three parts: `cipher-chainmode-IVmode`. So here cipher is AES, chainmode is CBC, and IV mode is essiv:sha256.

Consult the Linux documentation for more information at:

https://wiki.archlinux.org/index.php/Disk_encryption#Ciphers_and_modes_of_operation

8.7.5 Volume Encryption Key Management

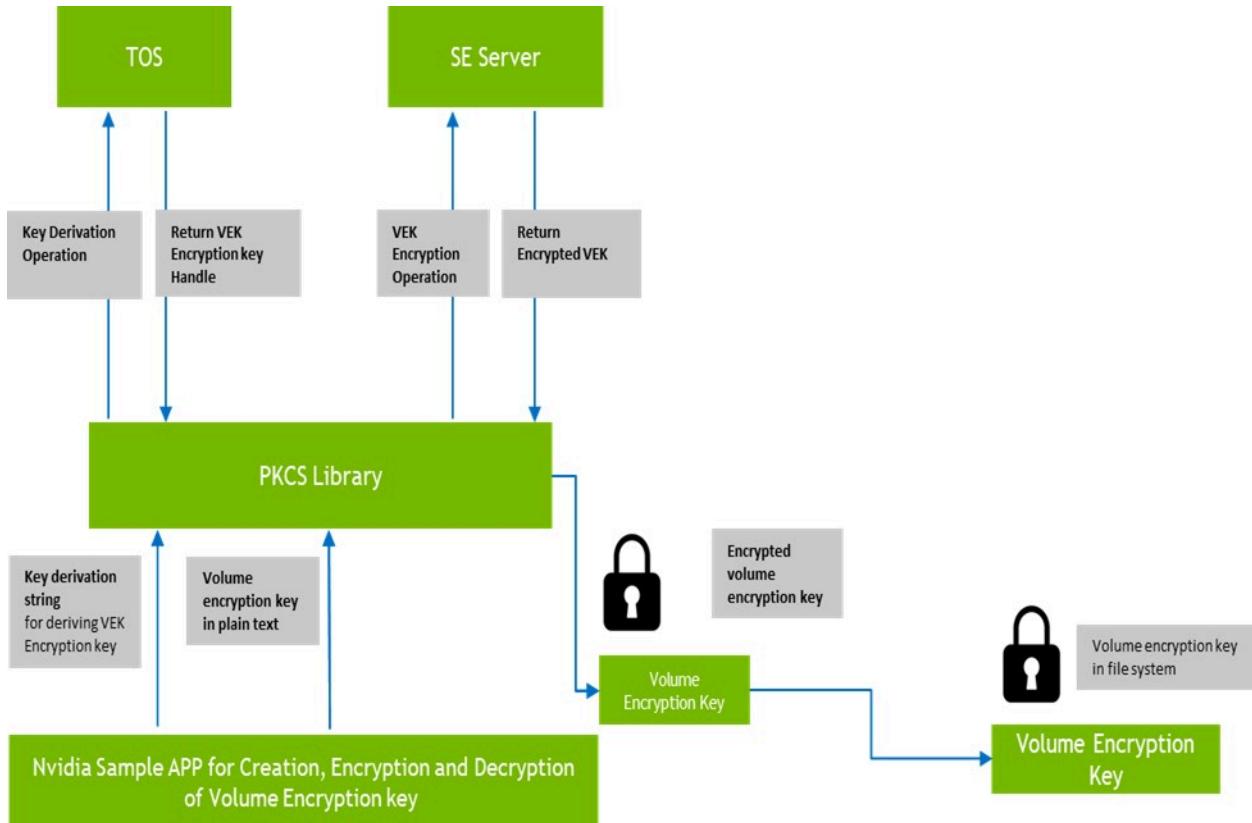
Different keys are involved in Encrypted File System (EFS) functionality:

- > Volume Encryption Key (VEK) : The Volume encryption key ensures the data flowing between the file system and the disk is encrypted and decrypted. It is passed as input to dmsetup tool which in turn passes it to kernel dm-crypt layer which uses it for encryption/decryption of data at block level. VEK is generated using HW RNG (Random Number Generator) via PKCS#11 app.
- > VEK Encryption Key : VEK Encryption key is used to encrypt and decrypt VEK. This key is derived from OEM_K1 fuse key using NIST SP800-108 Counter KDF (HMAC-SHA256) with unique derivation string and label inputs using PKCS#11 app. In later releases, VEK Encryption key will be derived from OEM_KDK0 instead of OEM_K1.

8.7.5.1 Encryption of VEK

The diagram below shows Encryption of VEK using PKCS#11 app. It includes following steps:

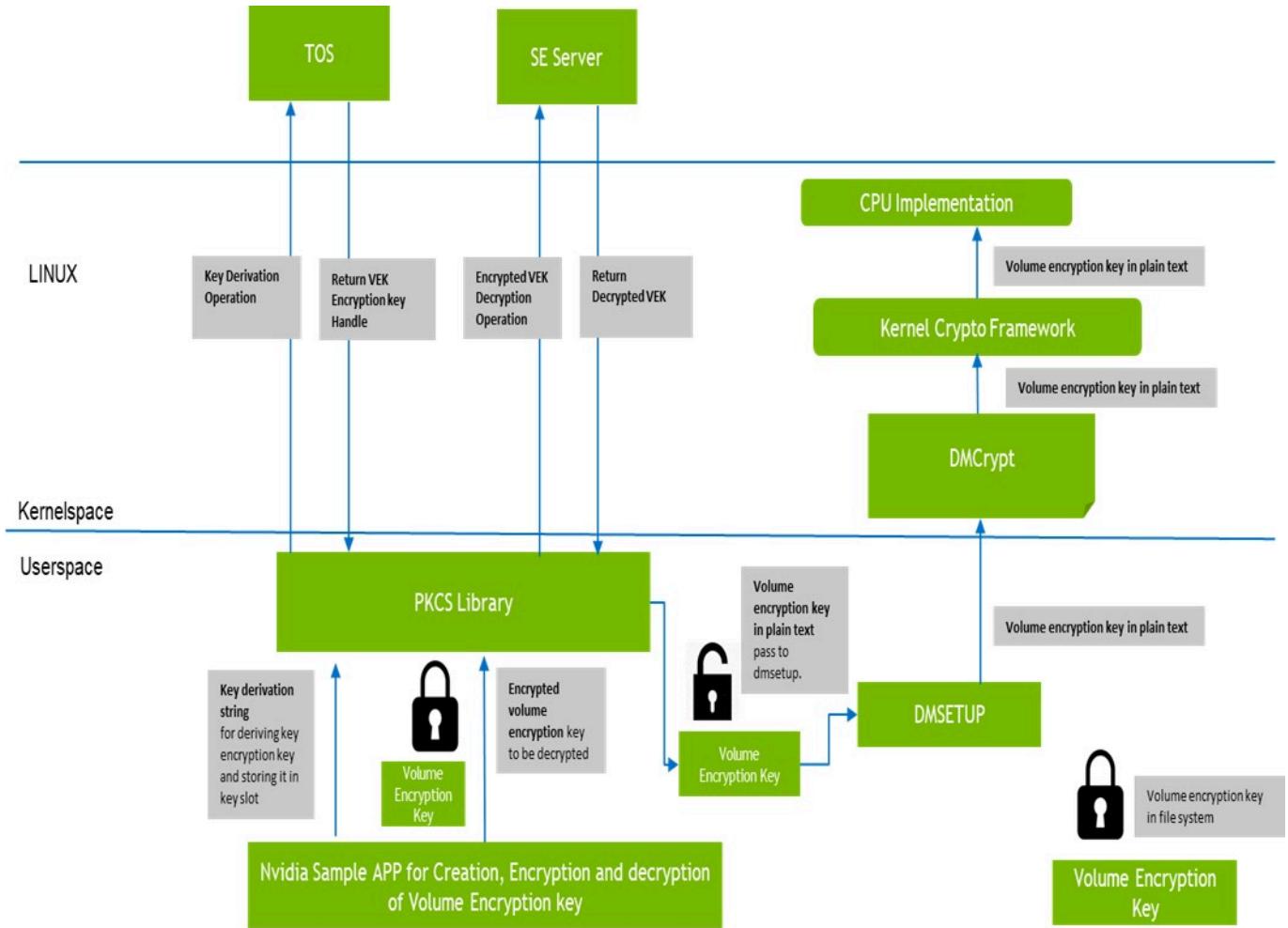
1. App passes VEK and Key derivation Strings to PKCS#11 library via their APIs.
2. PKCS#11 Library talks to TOS to derive VEK Encryption key based on key derivation Strings input.
3. PKCS#11 Library talks to SE Server to encrypt the VEK and returns Encrypted VEK.
4. App stores the Encrypted VEK in the filesystem (/etc/nvidia/efs/)



8.7.5.1.1 Decryption of VEK and Use by dm-crypt

The diagram below shows decryption of VEK and its use by dm-crypt via dmsetup. It includes following steps:

1. App reads Encrypted VEK from the filesystem (/etc/nvidia/efs/)
2. App passes Encrypted VEK and Key derivation Strings to PKCS#11 library via their APIs.
3. PKCS#11 Library talks to TOS to derive VEK Encryption key based on key derivation Strings input.
4. PKCS#11 Library talks to SE Server to decrypt the Encrypted VEK and stores the decrypted VEK in the file passed as input (/tmp/*).
5. EFS Systemd service reads the decrypted VEK from file in /tmp
6. EFS Systemd service will pass decrypted VEK as input to dmsetup which in turn is passed to kernel dm-crypt module for disk encryption and decryption operations.



8.7.5.2 Building Sample PKCS#11 App for VEK Generation, Encryption, and Decryption

The platform contains sample PKCS#11 app, which is used for:

- > Generation, encryption, and decryption of VEK.
- > Derivation of VEK encryption key from OEM_K1 with derivation strings as input.

The sources, headers, and build scripts are available at:

```
drive-linux/samples/security/efs_key
```

To build efs_key

- > Execute the command:

```
$ make
```

This generates efs_key executable.

To rebuild efs_key

- Execute these commands for incremental builds:

```
$ cd drive-linux/samples/security/efs_key
$ make
```

- Execute these commands for clean builds:

```
$ cd drive-linux/samples/security/efs_key
$ make clean
$ make
```

8.7.5.3 EFS Sample App Command Line Usage

The table below lists the different options for EFS sample app:

Short Option	Long Option	Description
-e	--encrypt	Encrypt VEK using derived key. This will also include VEK generation.
-d <filename>	--decrypt=<filename>	Decrypt VEK using derived key, writing the output to <filename>. <filename> must be a regular file under /tmp/ with mode 0600 and size 0. File must be present before passing it as input.
-p <string>	--derivation-string=<string>	Specify key derivation string (required, max 32 bytes). Used for deriving VEK Encryption Key from OEM_K1.
-c <string>	--context-string=<string>	Specify context string (optional, max 32 bytes. Default: context). Used for deriving VEK Encryption Key from OEM_K1.
-f <string>	--vek-name=<string>	Name suffix for filename storing key and IV (optional)
	--verbose	Print verbose information
-h	--help	Show usage

Example Usage for Creation and Encryption of VEK

```
./efs_key --encrypt --derivation-string=test --context-string=context --vek_name=homedir
```

The command above:

- Creates and encrypts VEK
- /etc/nvidia/efs/vek_homedir_enc.key file has the encrypted VEK and IV.

Example Usage for Decryption of VEK

```
./efs_key --decrypt=<tmp-input-file> --derivation-string=test --context-string=context --vek_name=homedir
```

The command above:

- > Decrypts the encrypted VEK from /etc/nvidia/efs/vek_homedir_enc.key
- > The decrypted VEK is written to <tmp-input-file> as a hexadecimal stream, one character per 4-bits.

Chapter 9. System Programming

This section provides guidance on adapting components to fit special needs. The tasks only apply if you modify and rebuild the kernel, boot loader, or device tree binaries.

9.1 Compiling the Kernel (Kernel 5.15)

This topic contains instructions for compiling the Linux kernel source in your Linux SDK product.

1. Set up the environment macros: The LOCAL_VERSION environment variable appends -rt-tegra to the kernel version and modules, please ensure \$NV_WORKSPACE is set to <top> of the PDK install directory.

```
export ARCH=arm64
export CROSS_COMPILE=${PWD}/toolchains/aarch64--glibc--stable-2022.03-1/bin/
aarch64-buildroot-linux-gnu-
export LOCALVERSION="-rt-tegra"
```

2. Set the kernel source directory as the current working directory:

```
cd $NV_WORKSPACE/drive-linux/kernel/source/oss_src
```

3. Enter the following command to apply the rt-patches.

```
cd kernel/scripts
bash generic-rt-patch.sh apply-patches
cd ../../
```

4. Install the packages to ready the kernel building workspace:

```
sudo apt-get update
sudo apt-get -f -y install imagemagick graphviz dvipng python3-venv fonts-noto-cjk latexmk librsvg2-bin texlive-xetex flex bison
```

5. Create an output directory and clean up.

```
mkdir out-linux
make -C kernel O=${PWD}/out-linux clean
```

6. Configure the kernel to the standard kernel with the command. To build a production kernel instead of a standard kernel, please use tegra_prod_defconfig instead of defconfig.

```
make -C kernel O=${PWD}/out-linux defconfig
```

7. Build the kernel:

```
make -j3 -C kernel O=${PWD}/out-linux
```



Note: If the preceding command fails, enter the `make` command without the `j<number>` option.

8. Build the MLNX Drivers using the steps below. MLNX Drivers build only applies to standard kernel defconfig (i.e., not the production or recovery kernel defconfigs).

```
cd $NV_WORKSPACE/drive-linux/kernel/source/oss_src/
mkdir -p out-linux/src-rt/
cp -al kernel ./out-linux/src-rt/
./kernel/scripts/build-module-mlnx.sh ./mlnx-drivers ./out-linux ./out-linux/src-rt/
kernel ./out-linux-mlnx-build
```

9. Build OOT modules:

```
ln -s -f ${PWD}/nvidia-oot /tmp/nv-oot
make -j3 -C ${PWD}/out-linux M=/tmp/nv-oot srctree.nvidia-oot=/tmp/nv-
oot srctree.nvidia=/tmp/nv-oot V=1 modules CONFIG_TEGRA_OOT_MODULE=m
CONFIG_TEGRA_VIRTUALIZATION=y
```



Note: If the preceding command fails, enter the `make` command without the `j<number>` option.

10. Consolidate the built kernel modules in the build directory with the following commands:

```
export INSTALL_MOD_PATH=${PWD}/out-linux
make -C kernel O=${PWD}/out-linux modules_install
make -C ${PWD}/out-linux M=/tmp/nv-oot modules_install
rm /tmp/nv-oot
```

11. Recompile the display kernel modules.

- a. Unify headers and `Module.symvers` for display module compilation:

```
cd $NV_WORKSPACE/drive-linux/kernel/source/oss_src
cat nvidia-oot/Module.symvers >> out-linux/Module.symvers
rsync -avzpq nvidia-oot/include/ out-linux/include
rsync -avzpq nvidia-oot/drivers/gpu/host1x/include/ out-linux/drivers/gpu/
host1x/include
```

- b. Untar the display source code.

The `NVIDIA-kernel-module-source-<Version>.tar.xz` source code is supplied as a `.xz` file. Untar the file. Its location is `<NV_WORKSPACE>/drive-linux_src/`

```
tar -xvf $NV_WORKSPACE/drive-linux_src/NVIDIA-kernel-module-source-
TempVersion.tar.xz
```

- c. Compile the display kernel module.

```
export IGNORE_PREEMPT_RT_PRESENCE=1
cd NVIDIA-kernel-module-source-TempVersion
make \
modules \
SYSSRC=$NV_WORKSPACE/drive-linux/kernel/source/oss_src/kernel \
SYSOUT=$NV_WORKSPACE/drive-linux/kernel/source/oss_src/out-linux \
SYSSRCHOST1X=$NV_WORKSPACE/drive-linux/kernel/source/oss_src/nvidia-oot/
drivers/gpu/host1x/include\
CC=${CROSS_COMPILE}gcc \
LD=${CROSS_COMPILE}ld.bfd \
```

```
AR=${CROSS_COMPILE}ar \
CXX=${CROSS_COMPILE}g++ \
OBJCOPY=${CROSS_COMPILE}objcopy \
TARGET_ARCH=aarch64 \
ARCH=arm64
```

- d. Copy the display modules to the module_install path.

```
mkdir -p $NV_WORKSPACE/drive-linux/kernel/source/oss_src/out-linux/lib/
modules/<kernel_version>/extra/opensrc-disp/
cd kernel-open
cp nvidia.ko nvidia-modeset.ko nvidia-drm.ko $NV_WORKSPACE/drive-linux/kernel/
source/oss_src/out-linux/lib/modules/<kernel_version>/extra/opensrc-disp/
```

- e. Set the kernel source directory as the current working directory.

```
cd $NV_WORKSPACE/drive-linux/kernel/source/oss_src
```

12. To flash the built kernel, update the kernel Image, kernel modules, and then the filesystem:

- a. Copy the uncompressed (Image) kernel images to the top of the kernel directory with the following command:

```
export PROD_SUFFIX=<str> # If using production kernel, set <str> to "_prod"
else set to empty string ""
sudo rm -fv $NV_WORKSPACE/drive-linux/kernel/preempt_rt${PROD_SUFFIX}/images/*
mkdir -p $NV_WORKSPACE/drive-linux/kernel/preempt_rt${PROD_SUFFIX}/images/
sudo cp -v ${PWD}/out-linux/arch/arm64/boot/Image ${PWD}/out-linux/vmlinux
${PWD}/out-linux/System.map $NV_WORKSPACE/drive-linux/kernel/preempt_rt
${PROD_SUFFIX}/images/
CAUTION: Before copying the new kernel images, back up the default kernels
provided.
```

- b. Copy the built modules to the SDK kernel modules path:

```
sudo rm -rf $NV_WORKSPACE/drive-linux/kernel/preempt_rt${PROD_SUFFIX}/modules/*
mkdir -p $NV_WORKSPACE/drive-linux/kernel/preempt_rt${PROD_SUFFIX}/modules/
sudo cp -a ${PWD}/out-linux/lib/modules/* $NV_WORKSPACE/drive-linux/kernel/
preempt_rt${PROD_SUFFIX}/modules/
sudo cp -a ${PWD}/out-linux-mlnx-build/mlnx-drivers-build/debian/mlnx-ofed-
kernel-modules/lib/modules/* $NV_WORKSPACE/drive-linux/kernel/preempt_rt
${PROD_SUFFIX}/modules/
```

- c. Copy the built modules to the root file system path with the following commands:

- i. To copy the updated modules into the root file system, use the following Build-FS steps:

- (1) Create a Build-FS JSON with the following content using the editor of your choice. Name it update_rfs.CONFIG.json and put it in \$PWD. The parameter <fstype> should be standard, production, production_debug depending on the list of modules to be copied to the filesystem.

```
{
    "OS": "linux",
    "Output": "driveos-updated-rfs",
    "Base": "${BASE_DIR}/targetfs.img",
    "FilesystemType": "<fstype>",
    "CopyTargets": [
        "${COPYTARGETYAML_DIR}/copytarget-kernel-modules.yaml"
    ],
}
```

```

        "PostInstalls": {
            "/etc/nvidia/run-once/nv-run-once-run-ldconfig": "target",
            "/etc/nvidia/run-once/nv-run-once-run-depmod": "target"
        }
    }
}

```

- (2) Execute Build-FS with the preceding configuration to rebuild the filesystem. Set NV_WORKSPACE to opt for SDK install directory (containing drive-linux directory).

```

export NVRTKERNELNAME="$(basename $NV_WORKSPACE/drive-linux/kernel/
preempt_rt${PROD_SUFFIX}/modules/*rt*-tegra)"
sudo -E /usr/bin/python3 -B /opt/nvidia/driveos/common/filesystems/
build-fs/17/bin/build_fs.py -w ${NV_WORKSPACE}/ -i $PWD/
update_rfs.CONFIG.json -o ${NV_WORKSPACE}/drive-linux/filesystem/
targetfs-images/
sudo rm -f ${NV_WORKSPACE}/drive-linux/filesystem/targetfs.img
sudo ln -s ${NV_WORKSPACE}/drive-linux/filesystem/targetfs-images/
driveos-updated-rfs.img ${NV_WORKSPACE}/drive-linux/filesystem/
targetfs.img

```

13. Update initramfs kernel modules with the ones built.

- a. If Building a production kernel, then please set

```
i. export INITRAMFS_IMAGE=$NV_WORKSPACE/drive-linux/filesystem/prod-
initramfs.cpio
```

- b. Else If building a standard kernel, please set

```
i. export INITRAMFS_IMAGE=$NV_WORKSPACE/drive-linux/filesystem/initramfs.cpio
```

- c. Extract the initramfs.

```
i. sudo rm -rf ./initramfs; sudo mkdir -p ./initramfs; cd ./initramfs
ii. sudo cpio -imdu --quiet < ${INITRAMFS_IMAGE}
```

- d. Run copytarget to copy kernel modules.

```

export PROD_SUFFIX=<str> # If using production kernel, please set <str> to
"_prod" else set to empty string ""
export NVRTKERNELNAME="$(basename $NV_WORKSPACE/drive-linux/kernel/preempt_rt
${PROD_SUFFIX}/modules/*rt*-tegra)"
export NV_SDK_NAME_LINUX=drive-linux
sudo rm -rf lib/modules/*
sudo -E /opt/nvidia/driveos/common/filesystems/copytarget/1/copytarget.py
$PWD $NV_WORKSPACE/ $NV_WORKSPACE/drive-linux/filesystem/copytarget/manifest/
copytarget-kernel-modules.yaml --filesystem-type boot_initramfs

```

- e. Re-create updated initramfs:

```
i. sudo find . | sudo cpio --quiet -H newc -o > ${INITRAMFS_IMAGE}
```

14. Flash the target using the SDK Bootburn tool to the board.

15. Boot up the board where the built kernel modules are auto-loaded on boot.

9.2 NVIDIA Supported Cross Toolchains

A cross toolchain refers to the compiler, linker, and target's C library that executes on the host (x86 or x86_64) but generates code for the ARM architecture. The C library is used for linking compiled code to create the target application.

The toolchains provided in the SDK(in \$TOP/toolchains) are described in the table below.

Directory	Components	Where to use the toolchain	Sources

Directory	Components	Where to use the toolchain	Sources
armv7-eabihf-stable-2020.08-1	GCC - 9.3.0 Binutils - 2.31 glibc-2.33 STDC++: 6.0.28 (GLIBCXX_3.4.28) Origin - Bootlin	Building 32-bit modules for trusted OS	<p>Debian Installer: nv-driveos-foundation-oss-src-<release><GCID>_amd64.deb</p> <p>Files: <top>/drive-foundation_src/bootlin/toolchain-sources_stable-2020.08-1_src.tar.gz</p> <p>Sources from upstream: https://toolchains.bootlin.com/downloads/releases/sources/</p> <p>Buildroot deconfig: Extract <top>/drive-foundation_src/bootlin/toolchain-sources_stable-2020.08-1_src.tar.<top>/drive-foundation_src/bootlin/toolchain-sources_stable-2020.08-1_src/armv7-eabihf-glibc-stable-2020.08-1.defconfig </p>
aarch64-stable-2022.03-1	GCC - 9.3.0 Binutils - 2.31 glibc-2.33 STDC++: 6.0.28 (GLIBCXX_3.4.28) Origin - Bootlin	Binding a guest PCT to the hypervisor, building 64-bit Quickboot, Building 64-bit Trusted OS, and building user-space components	<p>Debian Installer: nv-driveos-foundation-oss-src-<release><GCID>_<release><GCID>_amd64.deb</p> <p>Files: <top>/drive-foundation_src/bootlin/toolchain-sources_stable-2020.08-1_src.tar.gz</p> <p><top>/drive-foundation_src/bootlin/toolchain-sources_stable-2022.03-1_src.tar.gz</p> <p>Sources from upstream: https://10.72.0.6.0_v6.0.8.1/toolchains.bootlin.com/downloads/releases/sources/</p>

9.2.1 Steps to Rebuild the Toolchain

Obtain buildroot-toolchains_toolchains.bootlin.com-stable-2020.08-1.tar.gz and toolchain-sources_stable-2020.08-1_src.tar.gz after installing nv-driveos-foundation-oss-src-<release>_<GCID>_amd64.deb

The files are under <top>/drive-foundation_src/bootlin/

Common

1. `mkdir build`
2. `tar -C build -xpf toolchain-sources_stable-2020.08-1_src.tar.gz`

aarch64-glibc-stable-2022.03-1

1. `tar -C build -xpf buildroot-toolchains_toolchains.bootlin.com-stable-2022.03-1.tar.gz`
2. `tar -C build -xpf toolchain-sources_stable-2022.03-1_src.tar.gz`
3. Add the following lines to build/aarch64-glibc-stable-2022.03-1.defconfig:
 - a. `BR2_GCC_ENABLE_OPENMP=y`
 - b. `BR2_HOST_DIR="$(TOPDIR)/build"`
4. `cp build/aarch64-glibc-stable-2022.03-1.defconfig build/.config`
5. `cd build && make olddefconfig && make sdk && cd -`
6. Toolchain is available in `build/output/images/aarch64-buildroot-linux-gnu_sdk-buildroot.tar.gz`

armv5-eabi-glibc-stable-2020.08-1

1. `tar -C build -xpf buildroot-toolchains_toolchains.bootlin.com-stable-2020.08-1.tar.gz`
2. `cp build/armv5-eabi-glibc-stable-2020.08-1.defconfig build/.config`
3. `cd build && make olddefconfig && make sdk && cd -`
4. Toolchain is available in `build/output/images/armv5-eabi-buildroot-linux-gnu_sdk-buildroot.tar.gz`.

armv7-eabihf-glibc-stable-2020.08-1

1. `tar -C build -xpf buildroot-toolchains_toolchains.bootlin.com-stable-2020.08-1.tar.gz`
2. `cp build/armv7-eabihf-glibc-stable-2020.08-1.defconfig build/.config`
3. `cd build && make olddefconfig && make sdk && cd -`
4. Toolchain is available in `build/output/images/armv7-eabihf-buildroot-linux-gnu_sdk-buildroot.tar.gz`.

9.3 DRIVE OS Linux Yocto Components

This topic contains instructions on building the NVIDIA Yocto project-based components such as kernel, root file systems, and toolchains. With the Yocto Project, embedded Linux developers can build customized file systems that satisfy embedded constraints. The Yocto Project consolidates the efforts of the BitBake and OpenEmbedded communities.

Those projects enable users to easily define, configure, and cross-compile components required for embedded applications.

[®] NVIDIA DRIVE OS Linux provides the following Yocto root file system images:

- > Yocto-Drive OS AV (tegra-drive-os-av-image)
- > Cold-boot initramfs (tegra-initramfs-boot/tegra-knext-initramfs-boot/tegra-produce-knext-initramfs-boot)
- > Recovery initramfs (tegra-initramfs-recovery)

The Yocto DRIVE OS AV image consists of OSS components in accordance with Yocto 3.1 released baseline, NVIDIA proprietary platform enablement and acceleration components. This image is a suitable starting point for algorithm and application development for Autonomous Vehicle systems.

9.3.1 Building the Yocto Project Components for NVIDIA DRIVE Orin

9.3.1.1 Host Prerequisites

- > Ubuntu 20.04.2
- > Fast host CPU like Intel Core i5 or better (to reduce build time)
- > 50 GB Free space on HDD or more
- > 4GB RAM or more

The Yocto build is supported with two kinds of development kits, PDK and SDK. The SDK builds equivalently as the PDK, except for a few components, for which prebuilt are copied instead of compiled. To build DRIVE OS AV rootfs from the PDK, versions of PDK Debian files must be installed.

The Yocto build tool, bitbake, detects for the presence of the `version-nv-<pdk/sdk>.txt` file at `<top>/drive-linux/lib-target`, and builds the corresponding development kit.

9.3.1.2 Setting Up Linux Foundation and Compute Bits Using Debian Packages



Note: NVONLINE customers: Follow steps in the *NVIDIA DRIVE OS Installation Guide*, in the "Installation Using DRIVE OS Local Debian Packages" topic, to get the Debian packages.

Uninstall previously installed NVIDIA DRIVE OS versions to maintain integrity and to avoid any conflicts:

```
sudo -E apt-get -y --purge remove "nv-driveos*"
```

Install the following packages to set up NVIDIA DRIVE OS PDK for Yocto build:

```
sudo dpkg -i ./nv-driveos-repo-<SDK>-linux-[VERSION]-[GCID]_[VERSION]_amd64.deb
export NV_WORKSPACE=/path/where/SDK/needs/to/beinstalled
```

```
sudo -E apt -f -y install nv-driveos-build-<SDK>-linux-<VERSION>-><GCID> nv-driveos-
linux-oss-src-<VERSION>-<GCID>
nv-driveos-linux-yocto-<VERSION>-<GCID>
nv-driveos-linux-yocto-oss-src-<VERSION>-<GCID>
nv-driveos-foundation-oss-src-<VERSION>-<GCID>
nv-driveos-linux-tegra2aurix-updater-<VERSION>-<GCID>
```

Installing and Setting Up the Toolkits



Note: CUDA, CuDNN, TensorRT, and DriveWorks packages for Yocto are distributed through NVONLINE. Contact your NVIDIA representative for access to these packages.

Follow these steps to install and set up the toolkits:

1. [Install CUDA Toolkit on the Host](#).
2. [Install CuDNN Packages if Including cuDNN from Yocto](#).
3. [Install TensorRT Packages if Including TensorRT from Yocto](#).
4. [Install NVIDIA DriveWorks if Including NVIDIA DriveWorks from Yocto](#).

Install CUDA Toolkit on the Host



Note: This step is mandatory if CUDA is enabled with Yocto. CUDA toolkit must be installed on the host before Yocto build initialization. Skip this section if CUDA toolkit is already installed.

1. Follow the installation steps described in [Installing CUDA Debian Packages](#) under.
2. Copy the CUDA arm64 Debian package into a specific path as follows:

```
mkdir <top>/drive-linux/toolkits/cuda/drive-os-av/
cp cuda-tegra-repo-ubuntu*_arm64.deb <top>/drive-linux/toolkits/cuda/drive-os-av/
```

Install CuDNN Packages if Including cuDNN from Yocto

1. Follow the installation steps described in [Installing cuDNN Debian Packages](#).
2. Copy the cuDNN deb file into a specific path:

```
mkdir <top>/drive-linux/toolkits/cudnn/
cp cudnn-local-tegra-repo-ubuntu2004-<CUDNN_MAJOR_VER>-<CUDNN_MINOR_VER>-
local_1.0-1_arm64.deb <top>/drive-linux/toolkits/cudnn
```

Install TensorRT Packages if Including TensorRT from Yocto

1. Follow the installation steps described in [Installing TensorRT Debian Packages](#). Additionally, install tensorrt-safe-cross-aarch64 package if DriveWorks is enabled.



Note: Install the tensorrt-safe-cross-aarch64 package if DriveWorks is enabled.

2. Copy the TensorRT deb file into a specific path:

```
mkdir -p <top>/drive-linux/toolkits/tensorRT
cp nv-tensorrt-repo-ubuntu2004-cuda${CUDA_VER}trt${TRT_VER}-d61-target-
ga<BUILD_ID>-1_arm64.deb <top>/drive-linux/toolkits/tensorRT
```

Install NVIDIA DriveWorks if Including NVIDIA DriveWorks from Yocto



Note:

Before including NVIDIA DriveWorks in Yocto builds, you must install CUDA, CuDNN, and TensorRT toolkits and set up the respective Debian packages on the host as described in the preceding sections.



Note: Debian packages are only available to NVONLINE customers who have the NVIDIA DRIVE® OS PDK.

- Follow the installation steps:

```
sudo apt install -y ./driveworks_<dw-ver>-<GCID>_amd64.deb
sudo apt install -y ./driveworks-data_<dw-ver>-<GCID>_all.deb
sudo apt install -y ./driveworks-cross_<dw-ver>~linux<release-ver><GCID>_amd64.deb
sudo apt install -y ./driveworks-samples_<dw-ver>-<GCID>_amd64.deb
sudo apt install -y ./driveworks-stm_<dw-ver>-<GCID>_amd64.deb
sudo apt install -y ./driveworks-stm-cross_<dw-ver>~linux<release-ver>-<GCID>_amd64.deb
sudo apt install -y ./driveworks-stm-samples_<dw-ver>-<GCID>_amd64.deb
sudo apt install -y ./driveworks-cgf_<dw-ver>-<GCID>_amd64.deb
sudo apt install -y ./driveworks-cgf-data_<dw-ver>-<GCID>_all.deb
sudo apt install -y ./driveworks-cgf-cross_<dw-ver>~linux<release-ver>-<GCID>_amd64.deb
sudo apt install -y ./driveworks-cgf-samples_<dw-ver>-<GCID>_amd64.deb
```

- [OPTIONAL] Execute the steps below only if enabling STM and CGF with Yocto:

```
mkdir <top>/drive-linux/toolkits/driveworks
cp driveworks-cgf_<dw-ver>~linux<release-ver>-<GCID>_arm64.deb <top>/drive-linux/
toolkits/driveworks
cp driveworks-stm_<dw-ver>~linux<release-ver>-<GCID>_arm64.deb <top>/drive-linux/
toolkits/driveworks
```

For more information, see [To build NVIDIA Driveworks with tegra-drive-os-av-image](#).

9.3.1.3 Yocto DRIVE OS Linux Boot KPI



Note: This topic is applicable for NVONLINE customers only.

- Follow the instructions in the “Disabling Foundation Logs” and “Disabling BPMP Logs” sections in the DRIVE Linux Boot KPI chapter in the NVIDIA DRIVE OS 6.0 PDK Developer Guide.
- Modify <TOP>/drive-linux/kernel/source/oss_src/kernel/arch/arm64/configs/tegra_defconfig
- Replace CONFIG_DEBUG_FS=y with CONFIG_DEBUG_FS=m”
- Refer to [Building the Yocto Project Components for NVIDIA DRIVE Orin](#) and follow the steps to build Yocto.
- Flash the board following instructions in [Flashing Yocto Image and Running the Samples on Target](#).

6. When the board is flashed and successfully booted, on the target side replace the file `/usr/bin/nv_camera_display.sh` with the following:

For IMX728

```
#!/bin/bash

# Copyright (c) 2022, NVIDIA CORPORATION. All rights reserved.
#
# NVIDIA CORPORATION and its licensors retain all intellectual property
# and proprietary rights in and to this software, related documentation
# and any modifications thereto. Any use, reproduction, disclosure or
# distribution of this software and related documentation without an express
# license agreement from NVIDIA CORPORATION is strictly prohibited.

set -x

linux=$(uname -r);
display=false;

mkdir -m 0755 -p /root
mkdir -p /tmp
mkdir -p /dev/shm/

/sbin/insmod /lib/modules/$linux/kernel/drivers/platform/tegra/tegra_bootloader_debug.ko

echo "modprobe cdi_mgr cdi_tsc" > /sys/kernel/tegra_bootloader/add_profiler_record
/sbin/modprobe -a cdi_mgr cdi_tsc

echo "insmod nvmap.ko" > /sys/kernel/tegra_bootloader/add_profiler_record
/sbin/insmod /lib/modules/$linux/kernel/drivers/video/tegra/nvmap.ko

if [ "$display" = true ]; then
    echo "insmod nvidia.ko" > /sys/kernel/tegra_bootloader/add_profiler_record;/sbin/
insmod /lib/modules/$linux/extra/opensrc-disp/nvidia.ko rm_firmware_active="all";echo
"insmod nvidia-modeset.ko" > /sys/kernel/tegra_bootloader/add_profiler_record;/
sbin/insmod /lib/modules/$linux/extra/opensrc-disp/nvidia-modeset.ko;echo "insmod
nvidia-drm.ko" > /sys/kernel/tegra_bootloader/add_profiler_record;/sbin/insmod /
lib/modules/$linux/extra/opensrc-disp/nvidia-drm.ko modeset=1;echo "insmod nvgpu.ko"
> /sys/kernel/tegra_bootloader/add_profiler_record;/sbin/insmod /lib/modules/$linux/
kernel/drivers/gpu/nvgpu/nvgpu.ko;echo "launching app" > /sys/kernel/tegra_bootloader/
add_profiler_record;/usr/bin/nvsys_init_time -c "F008A120RM0A_CPHY_x4_s" -r 1 -d 1 --
disableISP2Output > /home/nvidia/nvsys_init_time_out.txt
else
    echo "launching app" > /sys/kernel/tegra_bootloader/add_profiler_record;/usr/bin/
nvsys_init_time -c "V1SIM728S1RU3120NB20_CPHY_x4" -m "0x0001 0x0000 0x0000 0x0000" -r 1
> /home/nvidia/nvsys_init_time_out.txt
fi

echo "starting systemd" > /sys/kernel/tegra_bootloader/add_profiler_record
exec /lib/systemd/systemd
```

For AR0820 Camera Config

```
#!/bin/bash
```

```

# Copyright (c) 2022, NVIDIA CORPORATION. All rights reserved.
#
# NVIDIA CORPORATION and its licensors retain all intellectual property
# and proprietary rights in and to this software, related documentation
# and any modifications thereto. Any use, reproduction, disclosure or
# distribution of this software and related documentation without an express
# license agreement from NVIDIA CORPORATION is strictly prohibited.

set -x

linux=$(uname -r);
display=false;

mkdir -m 0755 -p /root
mkdir -p /tmp
mkdir -p /dev/shm/

/sbin/insmod /lib/modules/$linux/kernel/drivers/platform/tegra/tegra_bootloader_debug.ko

echo "modprobe cdi_mgr cdi_tsc" > /sys/kernel/tegra_bootloader/add_profiler_record
/sbin/modprobe -a cdi_mgr cdi_tsc

echo "insmod nvmap.ko" > /sys/kernel/tegra_bootloader/add_profiler_record
/sbin/insmod /lib/modules/$linux/kernel/drivers/video/tegra/nvmap/nvmap.ko

if [ "$display" = true ]; then
    echo "insmod nvidia.ko" > /sys/kernel/tegra_bootloader/add_profiler_record;/sbin/
    insmod /lib/modules/$linux/extra/opensrc-disp/nvidia.ko rm_firmware_active="all";echo
    "insmod nvidia-modeset.ko" > /sys/kernel/tegra_bootloader/add_profiler_record;/
    sbin/insmod /lib/modules/$linux/extra/opensrc-disp/nvidia-modeset.ko;echo "insmod
    nvidia-drm.ko" > /sys/kernel/tegra_bootloader/add_profiler_record;/sbin/insmod /
    lib/modules/$linux/extra/opensrc-disp/nvidia-drm.ko modeset=1;echo "insmod nvgpu.ko"
    > /sys/kernel/tegra_bootloader/add_profiler_record;/sbin/insmod /lib/modules/$linux/
    kernel/drivers/gpu/nvgpu/nvgpu.ko;echo "launching app" > /sys/kernel/tegra_bootloader/
    add_profiler_record;/usr/bin/nvsys_init_time -c "F008A120RM0A_CPHY_x4_s" -r 1 -d 1 --
    disableISP2Output > /home/nvidia/nvsys_init_time_out.txt
else
    echo "launching app" > /sys/kernel/tegra_bootloader/add_profiler_record;/usr/bin/
    nvsys_init_time -c "F008A120RM0A_CPHY_x4_s" -r 1 > /home/nvidia/nvsys_init_time_out.txt
fi

echo "starting systemd" > /sys/kernel/tegra_bootloader/add_profiler_record
exec /lib/systemd/systemd

```

Create a Symbolic Link and Modify Permission

1. Create a symlink and modify file permissions to 777.

```

sudo ln -sf /usr/bin/nv_camera_display.sh /sbin/init
sudo chmod 777 /usr/bin/nv_camera_display.sh

```

2. Reboot the board. The nvsys_init_time application runs before systemd.

3. Run the following command to get the profiling data:

```

sudo cat /sys/kernel/tegra_bootloader/profiler

```

- Post where the boot KPI logs will be generated (in /var/log/syslog).

9.3.1.4 To build NVIDIA Yocto Project based components

- Install the dependent packages:



Note: Users making Yocto builds within the Docker image obtained from NGC should set the locale before launching bitbake.sudo locale-gen en_US en_US.UTF-8The Yocto build must be launched from a non-root userid, because Yocto does not support builds as root.

```
sudo su nvidia
```

- > Ensure the host system is connected to the Internet.
- > On the host, enter the following commands:

```
sudo add-apt-repository ppa:openjdk-r/ppa; sudo apt-get update; sudo apt-get install openjdk-8-jdk
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat xterm make xsltproc docbook-utils fop dblatex xmlto manpages-p1 manpages-fr-extra qemu-user libpulse-dev xxd python3-distutils
```

- Change to the build directory on the host system with the following command:

```
cd <top>/drive-linux_src/yocto
```

Where <top> is the directory on the host where the release is installed.

- Extract the contents of nvidia-layer.tgz and oss-packages.tgz with the following command:

```
tar xzf nvidia-layer.tgz
tar xzf oss-packages.tgz
```

- Export TEMPLATECONF to use templates from the meta-drive6 layer with the following command:

```
export TEMPLATECONF=$PWD/layers/meta-drive6/conf
```

- Initialize the Yocto project build environment with the following command:

```
source oss/dunfell/poky/oe-init-build-env
```

- Select the type of Image by setting the value of IMAGE_TYPE:

```
export IMAGE_TYPE=<full|minimal|samples|debug-utils>
```

- > minimal : Rootfs with upstream OSS (baseline Yocto distro) and Tegra SW libraries.
- > samples : Rootfs with contents of *minimal*, plus DRIVE OS samples (NVmedia, Gfx etc. samples).
- > debug-utils : Rootfs with contents of *minimal*, plus utilities to support debug and profiling (OSS and NV-proprietary debug utils).
- > full : Superset rootfs with everything in *minimal*, *samples* and *debug-utils* packaged.



Note: Default image type is set to *full*.

7. Enable network with Yocto build and download sources from upstream by setting BB_NO_NETWORK as follows:

```
export BB_NO_NETWORK='0'
```

8. Build component, image, or SDK/PDK depending on the recipe with the following command:

```
bitbake <recipe>
```

Recipes of interest are:

bitbake <recipe>	Function / Remarks
tegra-drive-os-av-image	Yocto DRIVE OS AV rootfs
tegra-initramfs-boot	Cold boot initramfs
tegra-prod-knext-initramfs-boot	Cold boot initramfs with production config
tegra-initramfs-recovery	Recovery initramfs (Not available for SDK customers. Only available for NVONLINE/PDK customers)

9. Kernel Security Feature Configurations:

- The default kernel recipe used with K5.15 utilizes prebuilt components to account for signature matching of the base kernel and Tegra kernel modules. To rebuild the base kernel from the source, launch bitbake on the linux-nvidia recipe manually
- SELinux support is enabled in Yocto with default mode as permissive. If you need to begin testing system behavior in enforcing mode, you must modify the DEFAULT_ENFORCING to "enforcing" in <top>/drive-linux_yocto/yocto/layers/meta-drive6/recipes-security/refpolicy/refpolicy-minimum%.bbappend.
- Steps to enable read-only root filesystem and dm-verity kernel security features are described in [DM-Verity and Read-Only File System Support](#)



Note: Recovery initramfs is a PDK feature. NVONLINE/PDK customers must ensure that the PDK packages are installed before launching bitbake for tegra-initramfs-recovery. SDK-only customers should ignore that recipe.



Note: To disable Auto IP Assignment (APIPA) on the target, append LinkLocalAddressing=no in the [Network] section of the /etc/systemd/networkd.conf file.

9.3.1.5 To build CUDA with tegra-drive-os-av-image



Note:

For the supported CUDA toolkit version, see the *Release Notes*.

Build CUDA <CUDA-toolkit-version> with Yocto DRIVE OS AV rootfs for DRIVE OS 6.0 platforms.

1. Export CUDA_ENABLE with <CUDA-toolkit-version>:

```
export CUDA_ENABLE=<CUDA-toolkit-version>
```

2. Build or rebuild the image:

```
bitbake tegra-drive-os-av-image
```

9.3.1.6 To build CuDNN with tegra-drive-os-av-image



Note:

For the supported CuDNN toolkit version, see product Release Notes.

Build CuDNN <CuDNN-toolkit-version> with Yocto DRIVE OS AV rootfs for DRIVE OS 6.0 platforms.

1. Export the CUDNN_ENABLE variable:

```
export CUDNN_ENABLE=<CuDNN-toolkit-version>
```

2. Build or rebuild the image:

```
bitbake tegra-drive-os-av-image
```



Note: The WARNING message "Unable to get checksum for cudnn-samples SRC_URI entry cudnnTest-<CUDA_VER>-ubuntu<>-aarch64-v<CUDNN_VER>.tgz: file could not be found" is expected when CUDNN_ENABLE is set.

9.3.1.7 To build TensorRT with tegra-drive-os-av-image



Note:

For the supported TensorRT toolkit version, see product Release Notes.

Build TensorRT <TensorRT-toolkit-version> with Yocto DRIVE OS AV rootfs for DRIVE OS 6.0 platforms.

1. Export the TENSORRT_ENABLE variable:

```
export TENSORRT_ENABLE=<TensorRT-toolkit-version>
```

2. Build or rebuild the image:

```
bitbake tegra-drive-os-av-image
```

9.3.1.8 To build NVIDIA Driveworks with tegra-drive-os-av-image



Note: Before building NVIDIA Driveworks with Yocto, you must install and enable CUDA, CuDNN, and TensorRT. For the supported NVIDIA Driveworks version, see the product Release Notes.

Build NVIDIA Driveworks <DriveWorks-version> with Yocto Drive OS AV rootfs for DRIVE OS 6.0 platforms.

1. Export the following toolkit variables:

```
export DW_ENABLE=<Driveworks-version> # eg export DW_ENABLE=5.2
export CUDA_ENABLE=<CUDA_VER> # pre-requisite
export CUDNN_ENABLE=<CUDNN_VER> # pre-requisite
export TENSORRT_ENABLE=<TensorRT_VER> # pre-requisite
```

2. Build or rebuild the image:

```
bitbake tegra-drive-os-av-image
```

9.3.2 Flashing Yocto Image and Running the Samples on Target

By default, bootburn does not pick up Yocto built, such as quickboot, kernel, and so on, for flashing due to known limitations.

9.3.2.1 To flash Yocto built Images via bootburn

These steps update the Yocto built images back to the release so that bootburn picks them up:

1. **Change to the Yocto deploy directory:**

```
cd <top>/drive-linux_src/yocto/build/tmp/deploy/images/${MACHINE}
```

2. **Copy the linux-DTBs to the release:**

```
mv <top>/drive-linux/kernel/preempt_rt <top>/drive-linux/kernel/preempt_rt_bkp
mkdir -p <top>/drive-linux/kernel/preempt_rt
cp *.dtb <top>/drive-linux/kernel/preempt_rt/
```

3. **Copy the kernel images and modules to the release:**

```
mkdir -p <top>/drive-linux/kernel/preempt_rt/images
cp Image <top>/drive-linux/kernel/preempt_rt/images/
cp vmlinux <top>/drive-linux/kernel/preempt_rt/images/
cp System.map <top>/drive-linux/kernel/preempt_rt/images/
mkdir -p <top>/drive-linux/kernel/preempt_rt/modules
# For standard kernel config :
cd <top>/drive-linux/kernel/preempt_rt/modules
tar xf <top>/drive-linux_src/yocto/build/tmp/deploy/images/${MACHINE}/modules-
${MACHINE}.tgz
find ./ -name '*rt-tegra*' -exec mv -t ./ {} +
# For production kernel config :
cd <top>/drive-linux/kernel/preempt_rt_prod/modules
tar xf <top>/drive-linux_src/yocto/build/tmp/deploy/images/${MACHINE}/modules-
${MACHINE}.tgz
find ./ -name '*rt-tegra*' -exec mv -t ./ {} +
```

4. **Copy the initramfs to the release:**

```
# For standard kernel config :
cp tegra-initramfs-boot-*.cpio <top>/drive-linux/filesystem/initramfs.cpio
# For production kernel config :
```

```
cp tegra-prod-initramfs-boot-*.cpio <top>/drive-linux/filesystem/prod-initramfs.cpio
```

5. **Copy the recovery initramfs to the release:**

```
cp tegra-initramfs-recovery-*.cpio <top>/drive-linux/filesystem/recovery-
initramfs.cpio
```

6. **Place the drive-os-av rootFS image at the location specified by the bootburn configurations:**

```
cd <top>/drive-linux/filesystem/targetfs-images
cp tegra-drive-os-av-image-${MACHINE}.img <top>/drive-linux/filesystem/targetfs-
images
ln -sf <top>/drive-linux/filesystem/targetfs-images/tegra-drive-os-av-image
${MACHINE}.img <top>/drive-linux/filesystem/targetfs.img
```

After following the steps above, all binaries are replaced by the Yocto built versions. Executing bootburn now automatically picks up these updated binaries. This applies to both native, as well as hypervisor flashing. For information on flashing steps, see the Flashing Customization topic in the *NVIDIA DRIVE OS 6.0 Linux SDK Developer Guide*.

9.3.2.2 Running Yocto built Apps on Target

1.  **Note:** Export LD_LIBRARY_PATH=/usr/local/cuda-<CUDA-VERSION>/targets/
 aarch64-linux/lib before running CUDA sample apps.

Run CUDA applications:

CUDA samples are packaged at /home/root/samples/cuda.

```
./sample-application>
```

2. Run TensorRT applications:

TensorRT samples are packaged at /home/root/samples/tensorrt

The following sample applications are available:

```
sample_char_rnn
sample_googlenet
sample_movielens
sample_fasterRCNN
```

sample_fasterRCNN requires the TensorRT model file to be present in the data/faster-rcnn/ subfolder. Download the model file here:

https://dl.dropboxusercontent.com/s/o6ii098bu51d139/faster_rcnn_models.tgz?dl=0

- 
- Note:** Export LD_LIBRARY_PATH=/usr/local/cuda-<CUDA-VERSION>/targets/
 aarch64-linux/lib before running CUDA sample apps.

3. Run the NVIDIA DriveWorks applications:

The NVIDIA DriveWorks 'core' samples are packaged at /usr/local/driveworks-<DW-VERSION>/bin.

```
./<sample-application>
```

If DriveWorks samples fail, export LD_LIBRARY_PATH:

```
export LD_LIBRARY_PATH=/usr/local/driveworks-<DW-VERSION>/lib:/usr/local/cuda-<CUDA-VERSION>/targets/aarch64-linux/lib
```



Note: The test data files required by some of the DriveWorks samples are not packaged in the rootfs by default, due to their large size. Samples from DW add-on packages are also not packaged.

In order to run the DriveWorks sample apps that require test data files, the scp utility may be used to push the required files into the target at /usr/local/driveworks-<DW-VERSION>/data/.

The additional DriveWorks samples may be pushed via scp onto the target, at /usr/local/driveworks-<DW-VERSION>/bin/ .

4. For running any OpenGL ES based applications on the target, kernel module nvidia_drm should be loaded by using insmod on target device:

```
insmod /lib/modules/<kernel_version>-rt63-tegra/extra/opensrc-disp/nvidia-drm.ko  
modeset=1
```

9.3.2.3 Setting the PATH for NVIDIA User

By default, bash-3.2 included with DRIVE-OS-AV rootfs excludes the following PATH settings for user nvidia:

```
/sbin  
/usr/sbin  
/usr/local/sbin
```

Since many utilities, such as ifconfig, are present under these directories, you must update the PATH setting include them by executing the following command on the target device:

```
# export PATH=$PATH:/sbin:/usr/sbin:/usr/local/sbin
```

Also add that command to the .bashrc for user nvidia, so that the updated settings are available in subsequent logins.

9.3.3 DM-Verity and Read-Only File System Support

The NVIDIA DRIVE® OS LINUX Yocto Cold-boot Initramfs (tegra-initramfs-boot) provides special support for DM-Verity when you want to flash and boot the Rootfs image whose integrity is verified and remains across system use. In DM-Verity workflow, the file system image content is validated on boot using the DM-verity root-hash and mounted read-only. DM-Verity workflow does not allow Rootfs to mount read/write as modifying the Rootfs partition changes its contents. The Initramfs enables Rootfs to write operations to go to a scratch partition to ensure normal operations (like logging). Both DM-Verity and handling of Rootfs write operations are detailed below.

Mounting Rootfs with DM-Verity Enabled

1. Without DM-Verity, the Rootfs ext4 image is flashed to the gos0-fs partition, which is mounted as read/write (unless explicitly set to read-only) in the PCT configuration.
2. With DM-Verity, the Rootfs ext4 is processed by NVIDIA DRIVE OS Bootburn to create the Rootfs Image having the ext4 image content with the verity header information appended. This content is flashed to the gos0-fs partition.
3. The Initramfs checks the kernel command line to see if the string `verity=1` is present. If `verity=1` is absent or `verity=0`, DM-Verity is disabled.
4. If the `verity=1` string is present, DM-Verity is enabled and proceeds to read the string starting from `verityinfo` and parses it to get the root-hash, root-hash offset, and the raw device containing the verity-enabled Rootfs image (by default, this is `/dev/vblkdev0p1`).
5. The Initramfs runs the cmd `veritysetup` with inputs: root-hash, root-hash offset, and the raw device containing the verity-enabled Rootfs image (for example, `/dev/vblkdev0p1`) to create the virtual plus mountable device: `/dev/mapper/vroot`.
6. `/dev/mapper/vroot` is mounted read-only and proceeds to prepare the scratch partition for Rootfs writes.

Using Scratch Partition for Rootfs Writes (Like Logging) When Rootfs Is Mounted Read-Only

The Rootfs can be mounted as read-only regardless of the state of DM-verity. If DM-Verity is enabled, the Rootfs must always be mounted as read-only. In this case, the Initramfs takes the following steps to use the scratch partition to enable Rootfs writes.

1. NVIDIA DRIVE OS PCT contains the writable gos-rw-overlay (by default, this is `/dev/vblkdev4`) partition of size 1 GB, and the mounted Rootfs (read-only) contains the directory `/rw_overlay`.
2. The device `/dev/vblkdev4` is mounted on `/rw_overlay` to create mount points for further mounts: `/rw_overlay/var`, `/rw_overlay/tmp`, `/rw_overlay/home`, and `/rw_overlay/etc`.
3. Finally, to route the Rootfs partition writes from `/tmp`, `/etc`, `/home`, and `/var` to `/rw_overlay`, Initramfs mounts as follows:
 - a. Mount overlayfs from `/rw_overlay/var` to `/var`.
 - b. Mount overlayfs from `/rw_overlay/etc` to `/etc`.
 - c. Mount overlayfs from `/rw_overlay/home` to `/home`.
 - i. Overlayfs mounts a, b, and c allow Rootfs to see the existing files in the respective directory, and route write operations to `/rw_overlay/*`.
 - d. Bind the mount directory `/rw_overlay/tmp` to `/tmp`.
 - i. In this case, `/tmp` in the Rootfs starts empty, and all read/write from `/tmp` to `/rw_overlay/tmp`.

9.4 Kernel Configuration

The Linux kernel used in this release is based on 5.15 major revision. Additionally, an optional package with 5.10 based kernel (which has been the primary kernel in prior releases) is shipped as well to help with the transition (though it has undergone minimal QA). The 6.0.7 release is the last release to support K5.10 kernel.

NVIDIA Linux PDK and SDK packages use the Linux kernel with PREEMPT-RT patches applied and the images shipped with the package are with PREEMPT-RT patches applied.

PREEMPT-RT patches can be obtained [here](#) and are applied on top of the Linux kernel. Additionally, some patches added by NVIDIA to resolve specific issues are also included.

The PREEMPT-RT patch files are available in the `kernel/rt-patches` folder in the Linux source (provided as a package as part of the release).

The primary change included with PREEMPT-RT patches is to convert spinlocks to mutex to make it preemptable.

The benefit of using PREEMPT-RT patches includes enabling real-time capabilities and reducing system latency, which are critical for NVIDIA DRIVE OS based products.

There may be slight degradation in performance in some cases with the kernel with PREEMPT-RT patches compared to the kernel without PREEMPT-RT patches.

For more information on PREEMPT-RT, see the [Linux Foundation](#) documentation.

Chapter 10. Bootloader Programming

The following sections describe bootloader programming.

DRAM Training Restriction

When creating DRAM DVFS tables using BPMP-FW-DTB, the BPMP-FW only supports hardware-based periodic DRAM training and frequencies higher than 1600Mhz. The BPMP-FW signals to HSM and halts boot if a software-based periodic training is needed for a DRAM DVFS table entry.

10.1 Understanding the Boot Flow Process

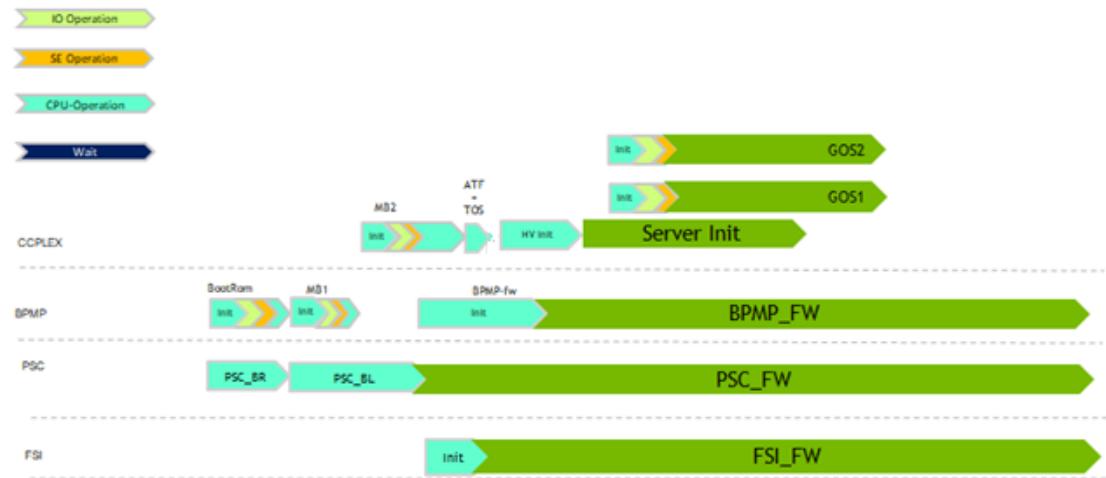


Note:

NVIDIA checks all binary integrity up to primary IFS image being loaded. Internal detail is not fully disclosed.

With Orin, the PSC processor handles all security (such as authentication/decryption) for BPMP_BR and MB1. Later in the process, PSC also owns all security keys for the system.

Upon power-up of the device, the boot flow sequence of events is as follows:



Notice that the BPMP processor runs:

- > BootROM
- > MB1
- > BPMP-FW

Notice that the CPLEX processor runs:

- > MB2

Notice that the PSC processor runs:

- > PSC-BootROM
- > PSC_BL
- > PSC-FW

10.1.1 Boot ROM

The boot ROM is hard-wired in the NVIDIA Orin chip to:

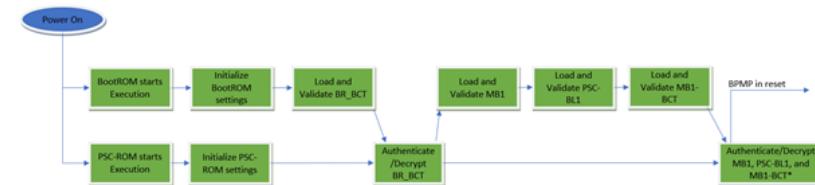
- > Initialize the necessary registers to access the desired boot media.
- > Load the boot components according to the boot sequence.

The initial boot medium is selected by using the strap resistor configuration BOOT-SELECT_CODE or by setting the RESERVED_SW and optionally BOOT_DEVICE_INFO fuse fields.

The PSC-ROM is also hard-wired in the NVIDIA Orin chip to:

- > Securely load the OEM keys from fuses and the NVIDIA keys from RTL into the Security Engine.
- > Authenticate and decrypt the binaries loaded by the boot ROM.

The early boot flow sequence is as follows:



Note: MB1-BCT is optionally decrypted depending on the BOOT_SECURITY_INFO fuse setting.

The boot ROM and PSC-ROM use the boot configuration table named BR_BCT, which contains information such as:

- > Storage location of BCH for MB1, PSC-BL1, and MB1-BCT
- > Boot chain parameters
- > Debug flags used by PSC-ROM
- > Validation

Verify the SHA-512 hash in the BCH/BCT that matches the computed value.

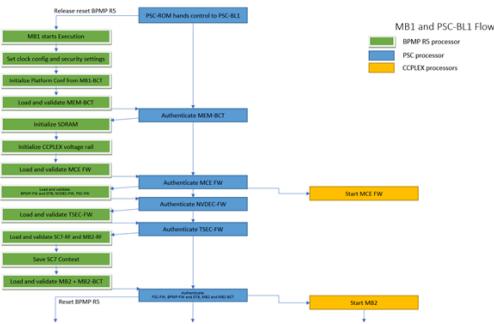
- > **Authentication**

Verify public signature by using the public key in the BCH/BCT, which is verified against its digest in fuses. The BCH contains the SHA-512, which is then validated again by PSC-ROM. For authentication and validation information, see [Understanding Security](#).

- > BR_BCT is not customer configurable except the customer_data fields.

10.1.2 Microboot 1 and PSC-BL1

After the BootROM has completed execution, PSC-ROM/PSC-BL1 releases reset on the BPMP R5 to start Microboot1 (MB1) in the boot flow process as follows:



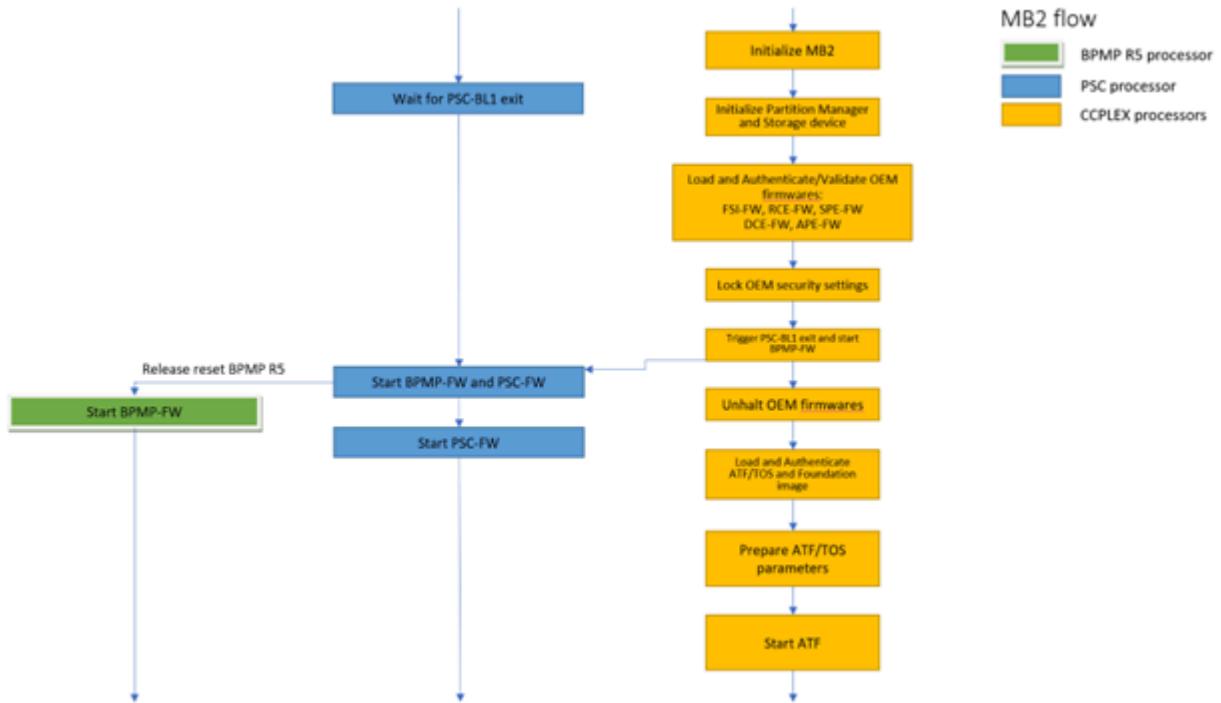
MB1 extends BootROM to provide the same security level in conjunction with PSC-BL1. During MB1 sequence, these tasks are executed:

- > Set clock and security settings
- > Initialize the platform configuration settings from MB1_BCT
- > Initialize the SDRAM based on the MB1 boot configuration table, MB1_BCT, and MEM_BCT.
- > Initialize the CCPLEX including MCE FW
- > Load/authenticate the NVDEC, BPMP-FW, PSC-FW and TSEC firmwares
- > Load the SC7 firmwares and prepare SC7 context
- > Load/authenticate

MB1_BCT is NOT customer configurable.

10.1.3 Microboot 2

The next stage of the boot sequence is Microboot 2 (MB2). The sequence is as follows:



In step 2, MB2 has ability to utilize QSPI, eMMC, or UFS as the boot device for the remaining boot payloads. This is determined through the NV Partition Table settings.

MB2 loads and validates additional OEM firmware that configures the SoC to run more complex software components. The additional firmware components are as follows:

Firmware Component	Description
Audio Processor Engine Firmware (APE-FW)	Provides the facilities for audio processing.
ATF/TOS image	Provide EL3 and Secure EL1 support.
Functional Safety Island Firmware (FSI-FW)	Provide the monitoring and reporting of functional safety errors.
Realtime Engine Firmware (RCE-FW)	Provides the realtime camera processing engine
SPE-FW	Provide debugging support in standard builds
Display Control Engine Firmware (DCE-FW)	Provides the realtime display processing engine.
Foundation image	Provides the Hypervisor stack and servers.

10.1.4 Authentication and Validation of Binaries

All firmware binaries and calibration data (i.e., BCTs, DTBs) loaded by BootROM, MB1, MB2, and Partition Loader are validated, authenticated, and optionally decrypted. The following category of binaries go through this process:

- > BPMP related firmware binaries including MB1, BPMP_FW, and associated calibration data
- > PSC related firmwares including PSC-BL1 and PSC-FW
- > CCPLEX related firmware binaries including MB2, MCE, ARM Trusted Firmware, Secure OS, partition table, and associated calibration data
- > CCPLEX virtualization binaries including Hypervisor, servers, Partition Loader, and associated calibration data
- > Auxiliary firmwares including FSI, DCE, RCE, PVA, APE
- > Key IST firmware binaries and calibration data
- > Runtime IST firmware binaries and calibration data
- > SC7 support binaries
- > Guest OS binaries including kernel, Primary IFS, and associated calibration data

A Binary Component Header (BCH) contains information about the binaries in a binary group. It is attached to the top of each binary or can be attached to one binary in the binary group. Up to four binaries can belong to a group. A BCH has an array of four elements, which contain:

- > Size of the binary
- > Version number
- > Hash value

For more information about BCH refer to [Grouping of Boot Images](#).

Dual Authentication of Firmware

MB1 firmware, BPMP firmware, and CPU firmware are delivered in binary form and are signed with the NVIDIA and OEM RSA keys. As a result, two validation steps are required:

1. Authentication and validation with the OEM key
2. Authentication and validation with the NVIDIA key

Recovery Support

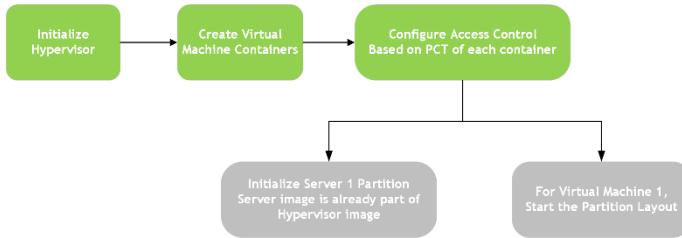
For authentication and validation information consult [Understanding Security](#).

SecureOS

The SecureOS (TOS) runs in EL3 mode and provides SecureMode support.

Hypervisor-based Flow

The hypervisor-based flow is as follows:



Virtual Machine

The Virtual Machine (VM) is an emulation of a native system. It is a software container that functions as an independent system with its own dedicated hardware and operating system, called guest OS in the context of Hypervisor.

Partition Loader

The PartitionLoader (PL) is a special purpose boot loader image embedded into the Hypervisor. It acts as Virtual boot ROM for the virtual machine.

10.1.5 Grouping of Boot Images

Authenticating each boot firmware binary to maintain chain of trust requires a great deal of time. Consequently, for boot time optimization, a new mechanism to group boot binaries is provided. A minimum of one, and up to a maximum of four, binaries can be made to form a group and refer to a single header as Boot Component Header (BCH) for a group. BCH has sha512 checksum for each binary in the group. BCH is prepended to the binary that is loaded first in that group.

During boot, BCH is authenticated and firmware binaries in the group are hash validated. This mechanism helps avoid authentication time for individual binaries.

For best practices and good design, consider these limitations when grouping the boot binaries:

1. Firmware loaded by the same loader can form a group.

For example:

- > Firmware loaded by the kernel, kernel-dtb and ramdisk can form a group.
- > Firmware loaded by MB2 i.e. bpmp-fw, bpmp-dtb, cpu-bootloader can form a group.
- > Firmware loaded by different loader cannot be part of a single group; the system cannot boot.

2. BCH must be prepended to the first binary loaded in the group. Consequently, the load order must be verified before grouping.

For example, if kernel, kernel-dtb and ramdisk are grouped together then BCH must be present on kernel.

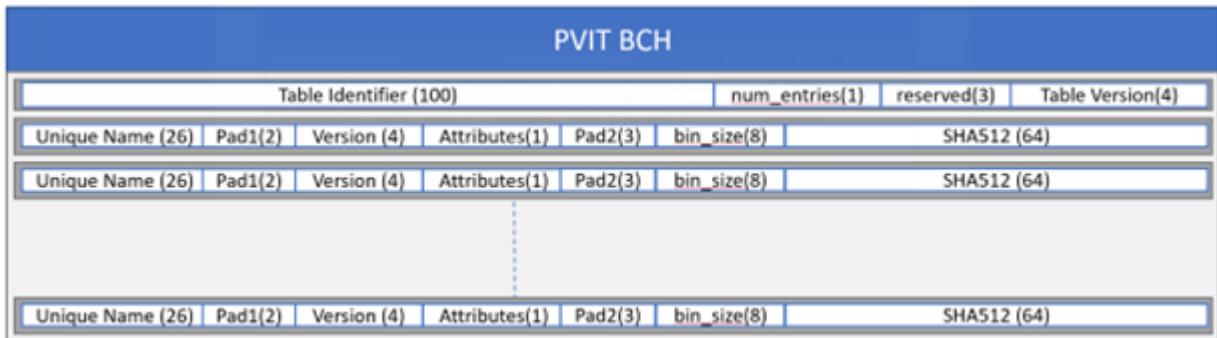
3. Boot image partition under a group must be updated together.
4. Grouping of NVIDIA signed binaries such as MB1, MTS-Preboot, MTS-BootPack cannot be changed by the OEM.

10.1.6 Grouping a Secure Boot Chain

Each boot chain when booted can be verified as a group of secure binaries in its entirety by enabling the partition version information table (PVIT) feature.

The PVIT is a structure, which contains the version and sha512 of each binary loaded in the normal boot flow. After the PVIT is securely loaded and authenticated using its associated Boot Component Header (BCH), each firmware is verified to match its corresponding entry's sha512 hash value as shown in the following table before execution or consumption of the data. The PVIT structure is passed on to the next stage of boot until the entire chain is verified.

The PVIT structure is defined as follows:



The PVIT feature is enabled in the MB2-BCT by setting the enable_pvิต field and adding the appropriate PVIT partition in the boot chain.

10.2 Using the Bootloader Recovery Mechanism



Note:

The default option is for GPIO boot chain select. When bf_bl_gpio_select_boot_chain_1b is enabled (=1), the emergency fail over does not work by default. If you want emergency failover from chain A to chain B on error, disable bf_bl_gpio_select_boot_chain_1b. If you want to use marker-based chain selection, then disable bf_bl_gpio_select_boot_chain_1b.

The bootloader includes:

- > BootROM
- > Microboot 1 (MB1)
- > Microboot 2 (MB2)
- > Quickboot (QB)
- > Hypervisor which includes:

- Partition Loader (PL)

These components load additional firmware components including:

- > Boot images
- > Partition images
- > Other firmware

The bootloader fails to load if:

- > Image corruption is declared: during boot, hash validation and signature authentication is performed. If the validation or authentication fails, the system declares that the image is corrupt.
- > Device read failure occurs during boot, if hardware issues are detected, the system returns a device read error.

These failures result in a boot process failure and therefore require using the provided bootloader recovery mechanism.

During the boot process, the bootloader recovery mechanism ensures a functioning firmware is loaded.

To ensure the recovery mechanism functions flawlessly, be aware of:

- > Firmware components have dependencies on each other.

For example, the BPMP firmware and kernel are dependent on each other. If the BPMP firmware version and the kernel version are functionally incompatible, the system functioning may be abnormal and operation may not be as expected.

- > Firmware updating process failures.

For example, if a power outage occurs while the firmware is being updated the BPMP firmware may be updated with the latest version while the kernel retains the outdated version. Due to this version mismatch, the system malfunctions and operation may not be as expected.

Therefore, redundant copies are a set of all the firmware components that are functionally compatible with each other. This set of firmware components is called Boot Chain.

- > The primary firmware components are on one boot chain.
- > Redundant firmware components are on another boot chain.

10.2.1 Recovery Mechanism Boot Chains

The recovery mechanism maintains up to three boot chains:

- > Boot Chain A
- > Boot Chain B
- > Boot Chain C

One bootchain is active at any given time.

- > The active boot chain is referred to as: Active Boot Chain.
- > The inactive boot chains are referred to as: Inactive Boot Chain.

10.2.1.1 Boot Chain Process

During normal operation, the bootloaders load the firmware components in the Active Boot Chain. If the system cannot boot the Active Boot Chain, the system resets to boot the other boot chain if marker-based boot chaining is enabled.

- > The BootROM, as a root of the boot chain, selects an initial Active Boot Chain.
- > Every bootloader must load firmware components from the Active Boot Chain.
- > If a bootloader fails to load a firmware component, the system switches the Inactive Boot Chain to the Active Boot Chain.

Advantages

The boot chaining process provides these advantages:

- > Handles cases of partial update so that the system is always bootable.
- > Except for a ratchet update case, each chain can be updated independently.
- > Compatibility issues between firmware components is eliminated.

Side Effects

A corrupted firmware component in each chain can cause an unusable system. For example, if using two boot chains and if the BPMP firmware in the Active Boot Chain is corrupted, and the kernel image in the Inactive Boot Chain is corrupted, the system is unable to boot any of the boot chains and cannot ever boot.

Components Outside the Boot Chain

Some firmware components are NOT included in any boot chain because of the nature of the components or due to BootROM limitation. For these components, multiple copies exist in the system. The boot loader locates the valid component from among the multiple copies.

- > The BootROM BCT selects the Active Boot Chain.
- > The Global Partition Table defines the images that belong to each boot chain.

Consequently, these components cannot belong to the boot chain.

10.2.2 Boot Recovery Mechanism Flow

Boot recovery implementation is as follows:

The data types used for the recovery mechanism include:

- > Scratch register
- > BootROM BCT
- > Soft fuses

10.2.2.1 Scratch Register

The PMC scratch register, SCRATCH_SECURE_RSV109_SCRATCH_0, also referred to as SCRATCHr, holds the Active Boot Chain and the Invalid Chain field.

The register bit definition is as follows:

Bit	Default	Setting	Description
31:6	0	RSVD	Do NOT modify this value. It is reserved.
5:4	X	ACTIVE_BOOT_CHAIN	When cleared to 0, the Active Boot Chain is set to 0, or Boot Chain A. When set to 1, the Active Boot Chain is set to 1, or Boot Chain B. When set to 2, the Active Boot Chain is set to 2, or Boot Chain C.
3	0	RSVD	Do NOT modify this value. It is reserved.
2	0	INVALID_CHAINC	When set to one, indicates chain is corrupted. When cleared to 0, indicates chain is not corrupt.
1	0	INVALID_CHAINB	When set to one, indicates chain is corrupted. When cleared to 0, indicates chain is not corrupt.
0	0	INVALID_CHAINA	When set to one, indicates chain is corrupted. When cleared to 0, indicates chain is not corrupt.

The contents of the scratch register are retained across soft reboots.

The scratch register is written when:

- > User wishes to boot a particular boot chain and the user writes the boot chain in the scratch register then issues a system reboot.
- > A bootloader detects corruption and the user updates the INVALID_CHAINx register bit and then sets a new Active Boot Chain.
- > Upon a normal cold boot, the BootROM initializes the register with the selected boot chain.

The contents of the scratch register are read when:

- > At any stage during boot, a bootloader reads the scratch register to find the Active Boot Chain. Once the firmware determines the Active Boot Chain, it loads the next stage firmware images in the chain.
- > Before beginning a system update, the Update tool reads the value to determine whether the system has booted the desired boot chain.

All bootloaders, except the hypervisor, have read and write permissions for the scratch register. Hypervisor access permissions for the scratch register are as follows:

- > Read access: The Partition Loader and Operating System Loader of each guest have read access to the scratch register. Additionally, the Monitor server has read access to the scratch register.
- > Write access: The Monitor server has write access to the scratch register. The PL, and Guest OS do NOT have write access.

10.2.2.2 BootROM BCT

The BootROM BCT includes the following data types to select the primary boot chain:

Data Type	Description
u32_non_gpio_select_boot_chain	<p>Indicates the primary boot chain when the GPIO selection is NOT enabled.</p> <p>0 = Boot Chain A 1 = Boot Chain B 2 = Boot Chain C Valid values are 0 to (u32_num_boot_chains – 1)</p>
bf_bl_gpio_select_boot_chain_1b	<p>Toggles to enable or disable the GPIO selection of the boot chain.</p> <ul style="list-style-type: none"> > 0 = disable GPIO selection and use the boot chain selected by u32_non_gpio_select_boot_chain_field. > 1= enable GPIO selection and use the boot chain selected by GPIO inputs. <p>SOC_GPIO31 controls lower bit for selection. SOC_GPIO41 controls upper bit for selection. Only used when u32_num_boot_chains > 2.</p>
u32_num_boot_chains	Indicates the number of boot chains in the system. Valid values are 1 – 3.

For guidance on connecting a GPIO to enable selection of the boot chain, refer to the Orin Interface Design Guide (DG-08535-001).

10.2.2.3 Soft Fuse

Soft fuses determine the recovery action to take when a bootloader fails to load a firmware component. The settings include:

Data Type	Description
SwitchBootChain	<p>If set, switches the boot chain.</p> <p>MB1 overwrites the value to 0 when GPIO selection is enabled.</p>
ResetToRecovery	<p>Used when the system does not switch the boot chain.</p> <p>When set to 1, the system reboots to forced recovery mode.</p> <p>When cleared to 0, the system hangs.</p>

10.2.2.4 Selecting the Active Boot Chain by BootROM

The BootROM, which is the root boot chain, is responsible for selecting the Active Boot Chain after the system powers up. The selection sequence is as follows:

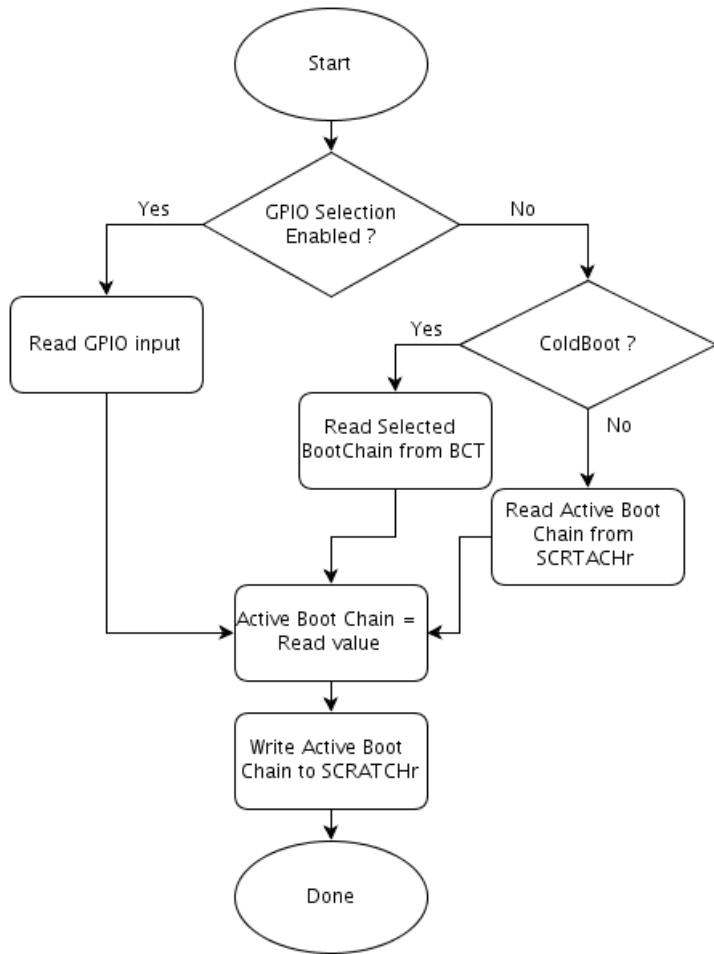
- Read `bf_b1_gpio_select_boot_chain_1b`, `u32_num_boot_chains` and `u32_non_gpio_select_boot_chain` information from BR BCT.
- If `bf_b1_gpio_select_boot_chain_1b` is enabled, read the value of `u32_num_boot_chains`.

If `u32_num_boot_chains` is less than 3, then read only `SOC_GPIO31` and use the input value as a Boot Chain. A GPIO value of 0 sets Boot Chain A as an Active Boot Chain, while a GPIO value of 1 sets Boot Chain B as an Active Boot Chain.

If `u32_num_boot_chains` is 3, then read `SOC_GPIO31` for bit 0 and `SOC_GPIO41` for bit 1 of chain selection. For GPIOs value of 00b sets Boot Chain A as an Active Boot Chain, for GPIOs value of 01b sets Boot Chain B as an Active Boot Chain, and for GPIOs value of 10b sets Boot Chain C as an Active Boot Chain.

- If `bf_b1_gpio_select_boot_chain_1b` is not enabled and it is a cold boot, BootROM uses `u32_non_gpio_select_boot_chain` value in BR BCT as an Active Boot Chain. Value 0 sets Boot Chain A, value of 1 sets Boot Chain B as Active Boot Chain, and value of 2 set Boot Chain C if supported.
- If `bf_b1_gpio_select_boot_chain_1b` is not enabled and it is not a cold boot, BootROM uses the active boot chain defined in the `SCRATCHr` register.

The flow is as follows:



10.2.2.5 Selecting the Boot Chain by the Loader

Each bootloader checks the contents of the SCRATCHr register to identify the Active Boot Chain. The bootloader then loads the next stage firmware components in the boot chain.

10.2.2.6 Triggering the Recovery Mechanism Inside a Guest OS Container

The recovery mechanism discussed under topic [Triggering Recovery Mechanism by Loader](#) is true until Hypervisor binary is loaded. Once Hypervisor boots up, it is no longer true due to the following reasons:

- > More than one guest OS is configured in the PCT. Each guest OS boots up independently in its guest OS environment provided by Hypervisor. Guest OSes do not have information about other guest OSes.
- > It may be possible that one or more guest OS boot fails, and other guest OSes boot up fine. There may be multiple boot failure scenarios here. How is each failure scenario handled?
- > If one or more guest OS boot fails, then how and who decides whether to trigger recovery mechanism or reboot that guest OS?

- > Triggering recovery mechanism in a guest OS environment involves informing Hypervisor. Hypervisor makes the final decision to trigger recovery.

To handle boot failures inside a guest OS environment, a different recovery mechanism policy is required inside the guest OS environment.

10.2.2.6.1 Scratch Registers

SCRATCH_SECURE_RSV109_SCRATCH_0: Bits 0-2 of this scratch register is called corrupt bit. It indicates whether inactive chain is good or corrupted. Value of 0 indicates inactive chain is good and 1 indicates inactive chain is corrupt.

10.2.2.6.2 Privileged Guest OS

The notion of privileged guest OS means that guest OS is allowed to read and write to the SCRATCH_SECURE_RSV109_SCRATCH_0 register. The request to read and write to the scratch register is sent to Hypervisor and Hypervisor in turn reads and writes to the physical scratch register. Privileged guest OS also has the ability to trigger a system reset. Hypervisor receives the system reset request and prepares the system for reboot.

10.2.2.6.3 Unprivileged Guest OS

This guest OS is also allowed to read and write to the SCRATCH_SECURE_RSV109_SCRATCH_0 register. The request to read and write is sent to Hypervisor and Hypervisor in turn does not read or write to the physical scratch register. Here, Hypervisor emulates the read and write. Unprivileged guest OS can request Hypervisor to perform a system reset but Hypervisor ignores the request and does nothing.

10.2.2.6.4 Marking a Guest OS as Privileged Guest OS

You must mark a guest OS as privileged guest OS in the PCT before flashing the system. Inside the PCT folder, in the `guest_config.h` file, set the `system_reset` attribute for a particular guest OS to mark it as privileged guest OS.

10.2.2.6.5 Assumptions

- > There must only be one privileged guest OS in the system.
- > System is booted using BR-BCT based boot chain selection mechanism. Recovery mechanism inside the guest OS environment does not work for GPIO based boot chain selection mechanism.

10.2.2.7 Triggering the Recovery Mechanism by BootROM

The BootROM loads MB1 from the Active Boot Chain.

- > If BootROM fails to load the MB1 image in the Active Boot Chain, the sequence is as follows:
 - Switch the Active Boot Chain and set the invalid boot chain to 1.
 - Load MB1 from the next new Active Boot Chain.

- If a failure occurs again, attempt to load next chain, if supported, or boot into forced recovery mode.
- > Soft fuse values are NOT used by BootROM.

Because BR BCT does NOT belong to any boot chain, recovery for this component takes place as follows:

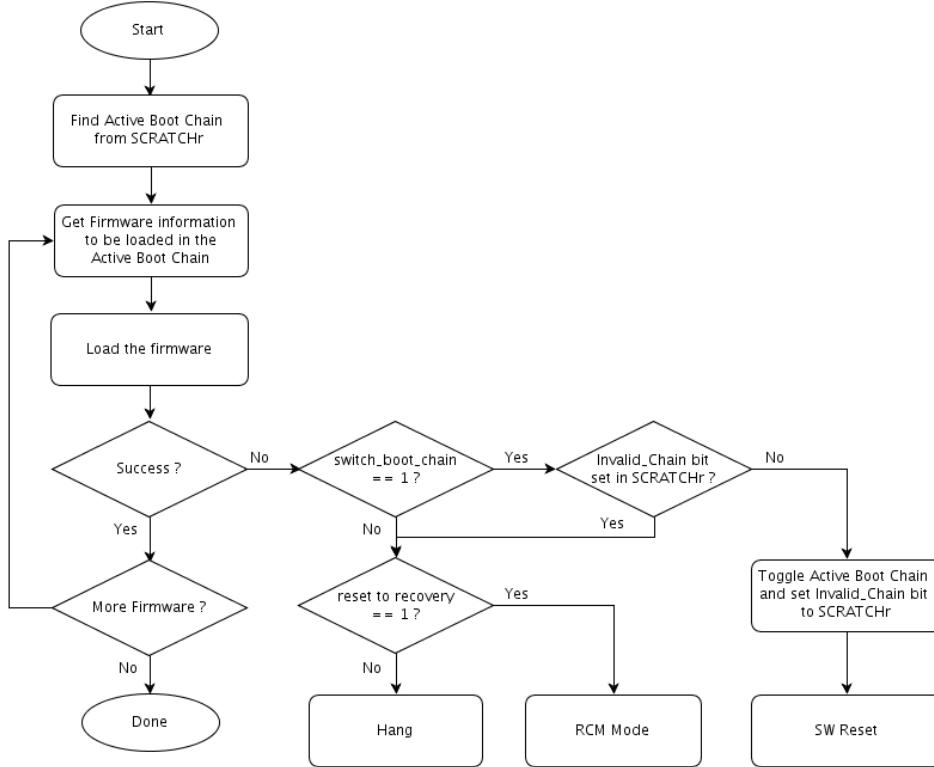
- > A single partition is provided for storing BR BCT.
- > The partition is shared by both boot chains. The partition contains multiple copies of BR BCT. All copies are identical.
- > Each copy is placed at the beginning of the boot storage device.
- > The BootROM handles the recovery of BR BCT. If one copy of BR BCT is found to be corrupted, the BootROM proceeds with the next copy of the BR BCT until a valid copy is located. If a valid copy is not located, BootROM resets into forced recovery mode.
- > At the time of the system update, each copy of BR BCT is updated to the new version to ensure all copies are at the same version.

10.2.2.8 Triggering Recovery Mechanism by Loader

During boot, when any bootloader, except BootROM, fails to load the next stage firmware, the recovery mechanism is triggered as follows:

- > Attempts to load the next stage firmware in the Active Boot Chain.
- > If the next stage firmware is loaded successfully, the loader continues to boot.
- > If the next stage firmware is NOT successfully loaded, the recovery mechanism is triggered.
- > If the INVALID_CHAINx bit in the SCRATCHr register is set to 1 or the switch_boot_chain soft fuse value is cleared to 0, these recovery actions are performed:
 - If the reset_to_recovery soft fuse value is set to 1, the system goes into forced recovery mode.
 - If the reset_to_recovery soft fuse value is cleared to 0, the system hangs.
- > If the INVALID_CHAINx bit is NOT set, and the switch_boot_chain soft fuse value is set to 1, then set the INVALID_CHAINx bit to 1 and change the ACTIVE_BOOT_CHAIN field in the SCRATCHr register and issue a reboot so that the system boots a different boot chain.

The flow for triggering the recovery mechanism by the loader is as follows:



MB2 and Quickboot load the Global Partition Table. Because this firmware component does not belong to any boot chain, the recovery flow is as follows:

- > There is a single partition to store the global partition table of the system.
- > The single partition contains multiple signed copies of the partition table. If one copy is corrupted, the system uses the next copy.
- > The global partition table contains information for both boot chains of the system. As a result, the global partition table must NOT be erased during the update. If the global partition table is erased, the system cannot be recovered without reflashing the entire images.

10.2.2.9 Partition Layout

The partition layout on flash is organized to support the recovery mechanism as follows:

- > For each partition, other than BR BCT and Global Partition Table, there are up to three partitions: Boot Chain A, Boot Chain B, and Boot Chain C.
- > For BR BCT, there is single common partition. This partition contains multiple copies of BR BCT residing at the beginning boot storage device.
- > For global partition table, there is single partition containing multiple copies of the partition table.

10.3 Ratcheting

There is a chance that the boot code may be updated due to a system flaw. It may also be possible that a security breach is identified in the software and that it is fixed in a newer version. The known vulnerabilities in the older versions of software can be leveraged by attackers to exploit the system security. Hence, the obsolete binaries must be prohibited from running on the chip.

Nonetheless, those antiquated binaries used to possess effective credentials to pass the secure boot authentication. Therefore, a version control mechanism is required to preclude the old binaries from passing the authentication so that these cannot be utilized by the attackers to exploit the system flaws. Rollback prevention checks, during the boot, the version of the software binaries against a known hardware version stored in a fuse and blocks the binaries from continuing if the software versions are found to be unacceptably old. Ratcheting is NVIDIA's rollback prevention mechanism.

The software checks for a monotonically increasing counter in the hardware that cannot be decreased once it is increased to a certain level. Ratcheting prevents binaries designed for an old chip from executing on a new chip. But the binaries designed for a new chip can run on an old chip to maintain the backward compatibility.

10.3.1 Software and Hardware Ratchet Versions

Ratcheting is achieved by maintaining a version. There is a software ratchet version and a hardware ratchet version.

10.3.1.1 Software Ratchet Version

Software has this version number maintained as a part of the binary. This version number is added to the signed section of generic header that is added at the beginning of the binary. This is used to find the ratchet level of the given binary without loading and booting it.

10.3.1.2 Hardware Ratchet Version

Hardware ratchet version is a monotonically increasing counter maintained on the chip. On the SoC, there are fuses reserved to hold this value.

10.3.2 Ratchet Constraints

- Let `SW_Ratchet_Version` denote the version of the software run on the chip.
- Let `HW_Ratchet_Version` denote the version corresponding to the ratchet value from fuses.

Ratcheting has the following constraints:

- > If `SW_Ratchet_Version < HW_Ratchet_Version` : Abort boot and trigger recovery flow. This case is considered as ratchet check failure.

Ratchet check failure results in either resetting to RCM or an alternate boot chain trigger based on softfuses in BR_BCT. See [Recovery Mechanism](#).

- > If `SW_Ratchet_Version = HW_Ratchet_Version` : Allow boot. The software is built exactly for that specific chip and, therefore, can run on the chip.
- > If `SW_Ratchet_Version > HW_Ratchet_Version` : Allow boot. Software is built for a higher version, so it has a required set of security patches applied already. In this case, it is running on an older chip, so allow it to run. In this case, updating the fuse is recommended.

Thus, older chips will continue to run new software if it is backward compatible. However, old software will not work on new chips/updated devices if the ratchet version on those devices is updated.

10.3.3 Opt-in Fuse

There is a dedicated fuse for opt-in. Based on its value, MB2 decides whether to perform in-field ratchet update or to skip it.

If OEM wants to opt-in for in-field ratchet update, opt-in fuse should be programmed to 1. The MB2 checks whether the opt-in fuse is burned before proceeding with the hardware ratchet update. Note that this check is software enforced only.

Opt-In Fuse Name	Bits
<code>FUSE_OPT_CUSTOMER_OPTIN_FUSE_0 [0]</code>	1 Bit

10.3.4 External Factors for Fuse Burning

The MB2 checks if the following conditions are favorable for fuse burning before actually burning the fuses.

- > VDD
- > VQPS
- > SoC Temperature

If all the above conditions are met, MB2 burns the fuses. If a condition is not met, it continues booting without burning the fuse and tries again on the next boot.

10.3.5 Software Components Protected by Ratcheting

The following software components are protected by ratcheting against any rollback attacks:

- > MB1
- > SC7 Firmware
- > MTS Firmware
- > Falcon Firmware
- > OEM Software Owned Components

10.3.6 Ratchet Levels for NVIDIA Owned Software Components

There are two sets of fuses dedicated for implementing the ratcheting for NVIDIA owned software components (i.e., NVIDIA owned ratchet fuse and OEM owned (or field) ratchet fuse).

The ratchet value for an NVIDIA owned software component is the sum of the ratchet value of NVIDIA owned ratchet fuse and OEM owned ratchet fuse.

10.3.6.1 NVIDIA Ratchet Level

- > The NVIDIA ratchet fuses are burned at the NVIDIA production line.
- > These fuses cannot be programmed from software.
- > For NVIDIA owned firmware, if any security loophole is found, then based on the criticality of the issue, the NVIDIA ratchet level can be updated.
- > An N bit NVIDIA ratchet fuse field can represent numeric value 0- 2^N-1 . (i.e., an absolute value is programmed in NVIDIA ratchet fuse).

10.3.6.2 OEM Ratchet Level

- > They are field programmable fuses and can be updated when the device is in the field.
- > This ratchet value is a thermometer encoded numerical value between 0 to N (i.e., number for bits set from LSB denote the fuse value).

10.3.6.3 Ratchet Level for OEM Owned Software Components

For implementing ratcheting for OEM owned software components, there are no NVIDIA owned ratchet fuses allocated. Only OEM owned ratchet fuses are allocated. The ratchet update happens in-field only.

10.3.6.4 OEM Firmware Ratchet Level

- > Only MB1-BCT is ratchet protected using hardware fuses.
- > Fuses are field programmable and can be updated when the device is in the field.
- > The ratchet value is thermometer encoded (a numerical value between 0 to N). For example, the number of bits set from LSB denotes the ratchet value.
- > Besides MB1-BCT, all OEM owned firmware loaded during boot are ratcheted using their absolute ratchet value in MB1-BCT.
- > OEM owned firmware ratchet versions are set in MB1-BCT and their corresponding BCH during flashing or offline image creation. For more information, see [Bootburn](#).

For more details, see [Ratchet Fuse Programming for OEM Owned Software Components](#).

10.3.7 Ratcheting for Falcon Firmware

Falcon firmware performs a self-enforced ratchet check.

10.3.7.1 Scratch Registers

There are no scratch registers allocated for storing the Falcon firmware ratchet version. Since the ratchet check for Falcon firmware is performed by the bootloader itself, not MB1, there is no requirement for the scratch register.

10.3.7.2 Fuses for Falcon Firmware

Fuse Ownership	Fuse Name	Bits	Bits	
NV Owned	FUSE_FALCON_UCODE_NV_REV [2:0]	3 Bits	TSECA	
	FUSE_FALCON_UCODE_NV_REV [4:3]	2 Bits	NVDEC	
	FUSE_FALCON_UCODE_NV_REV [6:5]	2 Bits	TSECB	
OEM Owned (Field)	FUSE_CCPLEX_UCODE_MB1_FA [15:0]	16 Bits	TSECA	T2_0
	FUSE_CCPLEX_UCODE_MB1_FA [31:16]	16 Bits	NVDEC	T2_0
	FUSE_CCPLEX_UCODE_MB1_FA [15:0]	16 Bits	TSECB	T3_0

10.3.8 Ratchet Fuse Programming for OEM Owned Software Components

- > MB1-BCT ratchet fuse programming follows the same sequences as defined for [NV Owned Firmware](#).
- > All OEM firmware (that are loaded during boot until guest ramdisk) have their ratchet version set in MB1-BCT and corresponding BCH set during flashing and during offline binaries generation. For more information about how to set the firmware ratchet version, see [Bootburn](#).

10.3.8.1 Scratch Registers

Software Component	Scratch Register	Bits
MB1-BCT	SECURE_RSV98_SCRATCH_0 [31:0]	32 Bits

10.3.8.2 Fuses for OEM Field Ratcheting

Fuse Ownership	Fuse Name	Bits
NV Owned	N/A	N/A
OEM Owned (Field)	fuse_system_fw_field_ratchet0[31:0]	32 Bits
	fuse_system_fw_field_ratchet1[31:0]	32 Bits
	fuse_system_fw_field_ratchet2[31:0]	32 Bits
	fuse_system_fw_field_ratchet3[31:0]	32 Bits

Passing Ratchet Status to Guest OSes

Ratchet fuse burning status is passed to guest OS via the kernel device tree. There are separate nodes for MB1, MTS, and MB1-BCT at the following location under the /proc interface.

```
/proc/device-tree/chosen/ratchet-status
```

Each node has two fields: status and error.

- > "error" has the appropriate ratchet error value.
- > "status" can have following status strings:

Ratcheting Status	Description
not_tried	Default status. Ratchet check path is skipped.
skipped_a	Active Boot Chain firmware ratchet matches with HW fuses.
skipped_b	Inactive Boot Chain firmware ratchet matches with HW fuses.
updated	Ratchet fuses are successfully updated with SW ratchet value.
failed	Ratchet fuse update(burning) failed.
no_option	Ratchet update check skipped as Opt-in fuse is not set by OEM

Lock Fuse Burning

If the SecurityMode fuse is burned, the quickboot locks fuse burning at the end of ratchet handling before kernel handoff to prevent malicious over-ratcheting.

Set bit #0 of register FUSE_DISABLEREGPROGRAM_0.

10.4 Ratchet Checks

10.4.1 Ratchet Check for NVIDIA Owned Software Components

For NVIDIA owned software binary, the true hardware ratchet level is obtained by taking sum of the NVIDIA owned ratchet level and OEM owned ratchet level.

This combined ratchet level is then compared with the software ratchet version of the binary.

The ratchet check is both self-enforced and loader enforced.

- > Self-enforced ratchet check:
 - This check resides within the same code that the rollback is trying to protect. In other words, the binary that is executing performs its own ratchet check.
 - It is basically a self-check to ensure that older binary does not continue execution on newer systems.
- > Loader enforced check:

- This check happens before the binary is even executed.
- The ratchet version of the binary being loaded is available as part of its boot component header, making the loader capable to fully investigate the rollback status.

If any ratchet check fails in Boot ROM, recovery mode is triggered.

If any ratchet check fails in MB1, then either boot with alternate chain is triggered or the target is put into recovery mode, depending on the soft fuses for boot chain options.

10.4.1.1 Ratchet Check for MB1

- > Boot ROM performs the loader enforced ratchet check for MB1.
- > MB1 performs a self-enforced ratchet check.

10.4.1.2 Ratchet Check for SC7 Firmware

- > Boot ROM and MB1 perform the loader enforced ratchet check for SC7.
- > SC7 performs a self-enforced ratchet check.

10.4.1.3 Ratchet Check for MTS Firmware

- > MB1 performs the loader enforced ratchet check for MTS.
- > MTS performs a self-enforced ratchet check.

10.4.1.4 Ratchet Check for Falcon Firmware

- > Falcon firmware performs a self-enforced ratchet check.
- > Loader enforced ratchet check for Falcon firmware is to be determined.

10.4.1.5 Ratchet Check for OEM Owned Software Components

- > There is no self-enforced ratchet check available for OEM owned software components.
- > The loader enforced ratchet check is performed by the corresponding loader of the binaries.
- > Apart from MB1-BCT, all OEM owned firmware ratchet levels are set in MB1-BCT and in the Binary Component Header (BCH) while flashing the target or generating binaries offline.
- > The loader for the binaries compares the ratchet value of the binary in BCH too see whether it is lower or equal to its ratchet value in MB1-BCT.
- > MB1-BCT is the only OEM owned firmware that is hardware ratcheted using hardware fuses dedicated for MB1-BCT.

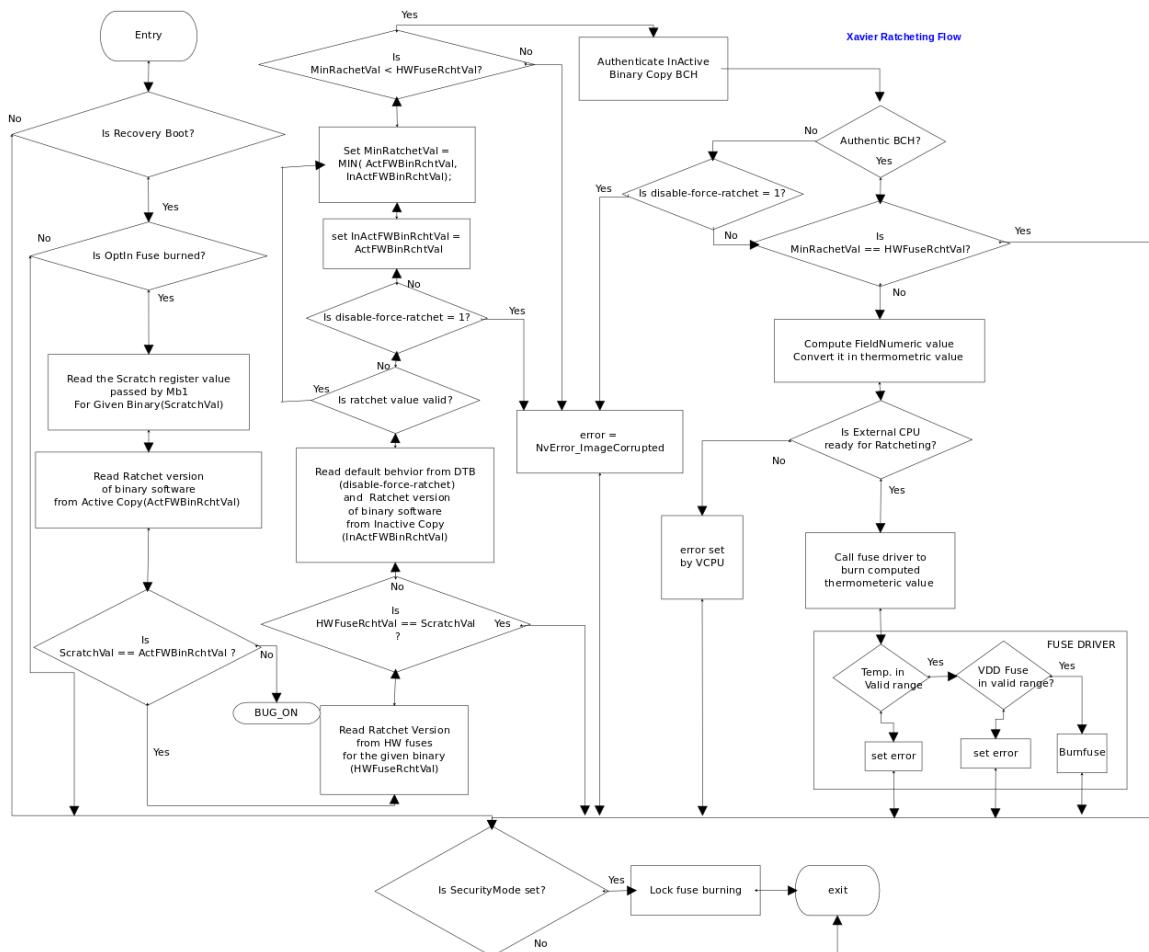
10.5 Rachet Fuse Programming

The following sections describe ratchet fuse programming for NVIDIA owned software components.

10.5.1 Ratcheting for MB1/SC7 and MTS Firmware

The overall flow:

- > MB1 passes the ratchet version of itself and SC7-FW to the bootloader via a scratch register dedicated for MB1 and SC-FW.
 - > MTS passes the OEM ratchet version of itself to the bootloader via a scratch register dedicated for MTS ratchet.
 - > The bootloader compares the ratchet versions of MB1, SC7, and MTS in their corresponding scratch register with respective hardware fuse values.
 - > Perform ratchet check and program the ratchet fuse.



For each of the binaries (MB1/SC7 and MTS), if ratchet version in scratch is greater than the hardware ratchet, then the minimum of the ratchet version of binary in boot chain A

and ratchet version of binary in boot chain B is programmed in the ratchet fuse, provided the following criteria are met:

- > [Opt-in fuse is burned.](#)
- > [External operating conditions are satisfied.](#)

The bootloader then passes the ratchet status to guest OS. For more information, see [Passing Ratchet Status to Guest OSes](#).

10.5.1.1 Scratch Registers

Software Component	Scratch Register	Bits
MB1	SECURE_RSV82_SCRATCH_0 [31:0]	32 Bits
SC7	SECURE_RSV98_SCRATCH_1 [31:0]	32 Bits
MTS	SECURE_RSV82_SCRATCH_1 [31:0]	32 Bits

10.5.1.2 Fuses for MB1/SC7

Fuse Ownership	Fuse Name	Bits
NVIDIA Owned	FUSE_MB1_NV_REV_0 [6:0]	7 Bits
OEM Owned (Field)	FUSE_CCPLEX_UCODE_MB1_FALCON_UC [31:0]	32 Bits

Note that there is only one fuse allocated for storing the ratchet of both MB1 and SC7.

10.5.1.3 Fuses for MTS

Fuse Ownership	Fuse Name	Bits
NVIDIA Owned	FUSE_CCPLEX_UCODE_NV_REV_0 [6:0]	7 Bits
OEM Owned (Field)	FUSE_CCPLEX_UCODE_MB1_FALCON_UC [31:0]	32 Bits

10.5.2 Ratcheting for Falcon Firmware

Falcon firmware performs a self-enforced ratchet check.

Scratch Registers

There are no scratch registers allocated for storing the Falcon firmware ratchet version. Since the ratchet check for Falcon firmware is performed by the bootloader itself, not MB1, there is no requirement for the scratch register.

Fuses for Falcon Firmware

Fuse Ownership	Fuse Name	Bits	Bits
NV Owned	FUSE_FALCON_UCODE_NV_REV_0 [2:0]	3 Bits	TSECA
	FUSE_FALCON_UCODE_NV_REV_0 [4:3]	2 Bits	NVDEC
	FUSE_FALCON_UCODE_NV_REV_0 [6:5]	2 Bits	TSECB
OEM Owned (Field)	FUSE_CCPLEX_UCODE_MB1_FALCON_UCODE FIELD_RATCHET2_0 [15:0]	16 Bits	TSECA
	FUSE_CCPLEX_UCODE_MB1_FALCON_UCODE FIELD_RATCHET2_0 [31:16]	16 Bits	NVDEC
	FUSE_CCPLEX_UCODE_MB1_FALCON_UCODE FIELD_RATCHET3_0 [15:0]	16 Bits	TSECB

10.6 Passing Ratchet Status to Guest OSes

Ratchet fuse burning status is passed to guest OS via the kernel device tree. There are separate nodes for MB1, MTS, and MB1-BCT at the following location under the /proc interface.

```
/proc/device-tree/chosen/ratchet-status
```

Each node has two fields: status and error.

- "error" has the appropriate ratchet error value.
- "status" can have following status strings:

Ratcheting Status	Description
not_tried	Default status. Ratchet check path is skipped.
skipped_a	Active Boot Chain firmware ratchet matches with HW fuses.
skipped_b	Inactive Boot Chain firmware ratchet matches with HW fuses.

Ratcheting Status	Description
updated	Ratchet fuses are successfully updated with SW ratchet value.
failed	Ratchet fuse update(burning) failed.
no_option	Ratchet update check skipped as Opt-in fuse is not set by OEM

10.7 Lock Fuse Burning

If the SecurityMode fuse is burned, the MB2 locks fuse burning at the end of ratchet handling before kernel handoff to prevent malicious over-ratcheting.

Set bit #0 of register FUSE_DISABLEREGPROGRAM_0.

10.8 Passing Customer Data to Guest OSes

Certain Customer Data, contained in the signed and unsigned sections of the BRBCT, is passed to guest OS via the kernel device tree. There are separate nodes for the supported BRBCT fields at the following location under the /proc interface.

/proc/device-tree/chosen/nvidia

Corresponding BRBCT field	Device Tree node
Serial ID	serialnumber
MAC_ID_INFO[n]	ether-mac<n>
SKU_INFO	sku
	sku_version
PROD_INFO	prodinfo
	prodver
BOARD_NAME	board_name
CHIP_EXT_INFO	chip_ext

10.9 Configuring MB1 Boot Configuration Table

Use these procedures to customize the Microboot 1 (MB1) Boot Configuration Table (BCT) with static platform-specific settings. The BCT settings that can be customized include:

- > SDRAM timing
- > Pinmux configuration
- > Security
- > Firmware configuration

During a system boot sequence, MB1 bootloader uses the MB1 BCT to configure platform-specific static settings. MB1 executes before other CPUs are enabled. The MB1 stage is owned by NVIDIA and signed by NVIDIA and the OEM.

For more information, see [Boot Flow](#).

10.9.1 Understanding the MB1 Boot Configuration Table

MB1 BCT specifies platform-specific data. When the bootburn script is called to flash a platform, it calls the tegrabct_v2 tool to create the MB1 BCT using:

- > Platform configuration files
- > tegral_mb1_bct.h header file

The BCT that is required by this stage is signed by the OEM. The MB1 stage performs platform specific initialization and sets up the secure control register (SCR).

For more information, see [Flashing with Bootburn](#).

The platform-specific configuration files specify:

- > Configuring the Pinmux and GPIO
- > Configuring the Prod Setting
- > Configuring the Pad Voltage Setting
- > Configuring the PMIC Setting
- > Configuring the Secure Register for BootROM

The configuration files are available at:

```
<top>/drive-foundation/platform-config/
t23x/automotive/bct/<board>/pinmux/
```

10.9.2 Configuring the Pinmux and GPIO

The pinmux configuration file provides pinmux and GPIO configuration information. The typical format for this data is register address and data, as a pair. MB1 allows writes to the pinmux and GPIO address range from the pinmux configuration table.

The pinmux configuration file is available at:

```
<top>/drive-foundation/platform-config/t23x/
automotive/bct/<board>/pinmux/tegra234-mb1-bct-pinmux-gpio-
<board>-<sku_id>-<rev>-t<a|b|c>.dtsi
```

For example:

```
tegra234-mb1-bct-pinmux-gpio-p3710-0001-b01-ta.dtsi
```

For information about converting the pinmux, GPIO, and pad DTS files to CFG format, see the Pinmux Generation Tool section in the *NVIDIA DRIVE OS6.0 Linux PDK Developer Guide*.

For definitions of the <board>-<sku_id>-<rev>-t<a|b|c>, see the *Release Notes*.

10.9.2.1 Usage

The command-line usage syntax is as follows:

```
pinmux.<address> = <value>;
```

Where:

- > pinmux is the domain name for GPIO and pinmux configuration data.
- > <address> is the absolute register address.
- > <value> is the 32-bit data value.

10.9.2.1.1 Device-side Implementation

The device side implementation is as follows:

```
write(value, address);
```

For example:

```
#### Pinmux for used pins ####
pinmux.0x02434060 = <value1>; # gen1_i2c_scl_pc5.PADCTL_CONN_GEN1_I2C_SCL_0
pinmux.0x02434064 = <value2>; # gen1_i2c_scl_pc5.PADCTL_CONN_CFG2TMC_GEN1_I2C_SCL_0
pinmux.0x02434068 = <value1>; # gen1_i2c_sda_pc6.PADCTL_CONN_GEN1_I2C_SDA_0
pinmux.0x0243406C = <value2>; # gen1_i2c_sda_pc6.PADCTL_CONN_CFG2TMC_GEN1_I2C_DA_0
      ::::
#### Pinmux for unused pins for low-power configuration ####
pinmux.0x02434040 = <value1>; # gpio_wan4_ph0.PADCTL_CONN_GPIO_WAN4_0
pinmux.0x02434044 = <value2>; # gpio_wan4_ph0.PADCTL_CONN_CFG2TMC_GPIO_WAN4_0
pinmux.0x02434048 = <value1>; # gpio_wan3_ph1.PADCTL_CONN_GPIO_WAN3_0
pinmux.0x0243404C = <value2>; # gpio_wan3_ph1.PADCTL_CONN_CFG2TMC_GPIO_WAN3_0
```

10.9.3 Configuring the Prod Setting

The prod setting provides the configuration of the settings for:

- > system characterization
- > Interface
- > Controller

These configuration settings are required to ensure the interface works in the platform. The prod setting is set at the controller level, and separately at the pinmux level. The examples provided describe the pinmux configuration.

The format of the configuration is a tuple of register address, mask, and data value. MB1 reads data from address, modifies it based on mask and value, and writes it back to address.

The prod configuration file is available at:

```
<top>/drive-foundation/platform-config/t23x/automotive
/bct/<board>/prod/tegra194-mb1-bct-prod-<board>-
<sku_id>-<rev>-t<a|b|c>.cfg
```

For example:

```
tegra194-mb1-bct-prod-e3550-0001-b01-ta.cfg
```

For definitions of the <board>-<sku_id>-<rev>-t<a|b|c>, see the *Release Notes*.

10.9.3.1 Usage

The command line usage syntax is as follows:

```
prod.<address>.<mask> = <value>;
```

Where:

- > prod is the domain name prefix for the setting
- > <address> is the pad control register address
- > <mask> is the mask value with 4 bytes, unsigned
- > <value> is the data value with 4 bytes, unsigned

10.9.3.1.1 Device-side Implementation

The device side implementation is as follows:

```
val = read(address)
val = (val & ~mask) | (value & mask);
write(val, address);
```

For example:

```
prod.0x02436010.0x00006000 = 0x00002000; # SDMMC4_DAT7,
DRV_TYPE: DRIVE_2X
prod.0x02436014.0x00006000 = 0x00002000; # SDMMC4_DAT6,
DRV_TYPE: DRIVE_2X
```

```

prod.0x02436018.0x00006000 = 0x00002000; # SDMMC4_DAT5,
DRV_TYPE: DRIVE_2X
prod.0x0243601c.0x00006000 = 0x00002000; # SDMMC4_DAT4,
DRV_TYPE: DRIVE_2X
prod.0x02436020.0x00006000 = 0x00002000; # SDMMC4_DAT3,
DRV_TYPE: DRIVE_2X

```

10.9.4 Configuring Pad Voltage Setting

Pins and pads support multiple voltage levels at a given interface. They can operate at 1.2 volts (V), 1.8 V or 3.3 V. Based on the interface and power tree of a given platform, the software must write the correct voltage of these pads to enable the interface. If pad voltage is higher than the Input/Output power rail, the pin does NOT function on that level. If pad voltage is lower than Input/Output power rail, the SOC pads can be damaged. Consequently, it is required to configure the correct pad voltage based on the power tree.

The pad configuration file is available at:

```
<top>/drive-foundation/platform-config/t23x/automotive/bct/<board>/padvoltage/tegra194-
mb1-bct-pad-<board>-<sku_id>-<rev>-t<a|b|c>.cfg
```

For example:

```
tegra194-mb1-padvoltage-e3550-0001-b01-ta.cfg
```

For definitions of the <board>-<sku_id>-<rev>-t<a|b|c>, see the *Release Notes*.

10.9.4.1 Usage

The command line usage syntax is as follows:

```
pad-voltage.<address> = <value>;
```

Where:

- > pad-voltage is the domain name prefix for the setting
- > <address> is the absolute register address
- > <value> is the 32-bit data value

10.9.4.1.1 Device-side implementation

The device side implementation is as follows:

```
write(value, address);
```

For example:

```

pad-voltage.0x0c36003c = 0x00000070; # PMC_IMPL_E_18V_PWR_0
pad-voltage.0x0c360040 = 0x00000053; # PMC_IMPL_E_33V_PWR_0

```

10.9.5 Configuring the PMIC Setting

During system boot, MB1 enables system power rails such as CPU, SRAM, and CORE as well as some system PMIC configurations. The typical configurations are:

- > Enabling rails
- > Setting rail voltages
- > FPS configurations

Enabling and setting of voltages of rails may require:

- > I2C command to devices
- > MMIO access to Tegra registers, either read-modify-write or write-only
- > Delay after the commands

Rail-specific configurations, such as I2C commands, MMIO access, and delays, are platform-specific. The MB1 BCT configuration file must provide configuration information.

The MB1-CFG format supports:

- > I2C commands and MMIO commands on any sequence.
- > Any I2C controller instance.
- > Any 7-bit secondary address of the device.
- > MMIO commands on read-modify-write format to support read only and Read-modify-write format.
- > I2C commands are read-modify-write format to support read only and Read-modify-write format.
- > Any amount of delay between commands.
- > Write only commands for I2C/MMIO.
- > Any size of device registers address and data size for i2c commands.
- > I2c command on the 400 KHz.
- > The sequence may be
 - 1 MMIO, 1 I2C
 - 1 I2C, 1 MMIO
 - 2 MMIO, 1 I2C
 - 1 MMIO, 2 I2C

The typical rail/configurations are divided into these PMIC command domains:

- > Generic: General PMIC configurations
- > GPU: Command related to CPU rails
- > GPU: Commands related to GPU
- > SRAM: Commands related to SRAM
- > CORE: Commands related to CORE
- > MEM: Commands related to Memory

If the configuration for given rail is NOT specified, it is not necessary to provide the command sequence of that rail. MB1 device side code ignores the configuration of that rail.

Each rail is defined with a unique ID to enable the parsing and BCT binary. The unique IDs are as follows:

Rail Name	ID
GENERIC	1
CPU	2
CORE	3
SRAM	4
GPU	5
MEM	6

The PMIC configuration file is available at:

```
<top>/drive-foundation/platform-config/t23x/
automotive/bct/<board>/pmic/tegra194-mb1-bct-pmic-
<board>-<sku_id>-<rev>-t<a|b|c>.cfg
```

For example:

```
tegra194-mb1-bct-pmic-e3550-0001-b01-ta.cfg
```

For definitions of the <board>-<sku_id>-<rev>-t<a|b|c>, see the *Release Notes*.

10.9.5.1 Usage

The command line usage syntax for the common parameters are as follows:

```
pmic.<parameter> = <value>;
```

The command line usage syntax for the rail-specific parameters are as follows:

```
pmic.<rail-names>.<rail-id>.<parameters> = <value>;
```

Where <parameters> is as follows:

Parameter	Description
command-retries-count	Specifies the number of allowed command attempts.
wait-before-start-bus-clear-us	Specifies the wait timeout, in microseconds, before issuing the bus clear command. The wait time is calculated as: $1 \ll n$ microseconds Where $i > n$ is as provided by this parameter.
rail-count	Specifies the number of rails in this configuration file that must be configured.

For example:

```
pmic.command-retries-count = <value>;
pmic.wait-before-start-bus-clear-us = <value>;
pmic.rail-count = <value>;
```

10.9.5.1.1 Rail-Specific Parameters

Rail-specific parameters take the following format:

- The rail specific commands are divided into blocks.
- Each rail can have one or more blocks. Each block of given rails are indexed starting from 0.
- Each block contains either MMIO or I2C commands. If both MMIO and I2C commands are required then commands are broken into multiple blocks.
- If block contains I2C type of commands then all commands are sent to same device. If it is require having i2c commands for multiple devices then it needs to split into multiple blocks.
- If commands on given blocks are I2C type the device address, register address size, register data size are parameters which is not needed for MMIO commands.
- Given block can contain more than one commands but all commands are same type.
- Delay is provided after each commands of a given blocks. The delay are same for all commands. If different delay are required then it need to split into multiple blocks.

The details for each parameter are as follows:

Parameter	Description
pmic.<rail-name>.<rail-id>	Specifies the rail specific parameters prefixes.

Parameter	Description
pmic.<rail-name>.<rail-id>.block-count = <value>;	Specifies the block count. Where <value> for the block count is the number of command blocks for a given rail.
pmic.<rail-name>.<rail-id>.block[index]	Specifies the block identification, where all blocks are indexed, starting from 0.
type	Specifies the command type. Available types include: <ul style="list-style-type: none"> > MMIO (0) > I2C (1)
delay	Specifies the delay, in microseconds, after each command in a given block.
count	Specifies the number of commands in a block.

10.9.5.1.2 I2C type-specific Parameters

The I2C type specific parameters are as follows:

Parameter	Description
I2C-controller-id	Specifies the controller ID of I2C.
slave-add	Provides the 7-bit secondary address.
reg-data-size	Specifies the register size in bits: 0 or 8:1 byte 16: 2 byte
reg-add-size	Specifies the register address size in bits: 0 or 8:1 byte 16: 2 byte

- > Commands can be MMIO or I2C.
- > The information is in the format <address>.<mask> = <data>, to support the read-modify-write sequence.
- > All commands are indexed, to facilitate multiple commands in a given block.

- > Commands are sent to device in sequence, starting from index 0.

The command syntax is as follows:

```
commands[command-index].<addr>.<mask> = <data>;
```

10.9.5.2 Generic Format

The code snippets show the common and rail specific parameters in a generic format.

The common parameters are:

```
pmic.command-retries-count = <u32>;
pmic.wait-before-start-bus-clear-us = <u32>;
pmic.rail-count = <u32>;
```

The rail-specific parameters are:

```
pmic.<rail-name>.block-count
```

The generic format is as follows:

```
##### BLOCK 0 #####
pmic.<rail-name>.<rail-id>.block[0].type = <0 for MMIO, 1 for I2C>
pmic.<rail-name>.<rail-id>.block[0].delay = <u32>
pmic.<rail-name>.<rail-id>.block[0].count = <calculated>;
#For I2C specific
pmic.<rail-name>.<rail-id>.block[0].I2c-controller-id = <u32>;
pmic.<rail-name>.<rail-id>.block[0].slave-add = <u32>;
pmic.<rail-name>.<rail-id>.block[0].reg-data-size = <u32>;
pmic.<rail-name>.<rail-id>.block[0].reg-add-size = <u32>;
#I2C and MMIOs
pmic.<rail-name>.<rail-id>.block[0].commands[0].<addr>.<mask> = <data>;
pmic.<rail-name>.<rail-id>.block[0].commands[1].<addr>.<mask> = <data>;
pmic.<rail-name>.<rail-id>.block[0].commands[2].<addr>.<mask> = <data>;
pmic.<rail-name>.<rail-id>.block[0].commands[3].<addr>.<mask> = <data0>;
::::
##### BLOCK 1 #####
pmic.<rail-name>.<rail-id>.block[1].type = <0 for MMIO, 1 for I2C>
pmic.<rail-name>.<rail-id>.block[1].delay = <u32>
pmic.<rail-name>.<rail-id>.block[1].count = <Calculated>
#For I2C
pmic.<rail-name>.<rail-id>.block[1].I2c-controller-id
pmic.<rail-name>.<rail-id>.block[1].slave-add
pmic.<rail-name>.<rail-id>.block[1].reg-data-size
pmic.<rail-name>.<rail-id>.block[1].reg-add-size
#I2C and MMIOs
pmic.<rail-name>.<rail-id>.block[1].commands[0].<addr>.<mask> = <data>;
pmic.<rail-name>.<rail-id>.block[1].commands[1].<addr>.<mask> = <data>;
pmic.<rail-name>.<rail-id>.block[1].commands[2].<addr>.<mask> = <data>;
pmic.<rail-name>.<rail-id>.block[1].commands[3].<addr>.<mask> = <data0>;
::::
```

For example, the usage is as follows:

```
pmic.command-retries-count = 1;
```

```

pmic.wait-before-start-bus-clear-us = 0;
pmic.rail-count = 6;
##### GENERIC RAIL (ID = 1) DATA #####
pmic.generic.1.block-count = 1;
# 1. Set PMIC MBLDP = 1, CNFGGLBL1 bit 6 = 1
pmic.generic.1.block[0].type = 1; # I2C Type
pmic.generic.1.block[0].i2c-controller-id = 4;
pmic.generic.1.block[0].slave-add = 0x78; # 7Bit:0x3c
pmic.generic.1.block[0].reg-data-size = 8;
pmic.generic.1.block[0].reg-add-size = 8;
pmic.generic.1.block[0].delay = 10;
pmic.generic.1.block[0].count = 1;
pmic.generic.1.block[0].commands[0].0x00.0x40 = 0x40;
##### CORE RAIL (ID = 3) DATA #####
pmic.core.3.block-count = 2;
# 1. Set 950mV voltage.
pmic.core.3.block[0].type = 1; # I2C Type
pmic.core.3.block[0].i2c-controller-id = 4;
pmic.core.3.block[0].slave-add = 0x70; # 7Bit:0x38
pmic.core.3.block[0].reg-data-size = 8;
pmic.core.3.block[0].reg-add-size = 8;
pmic.core.3.block[0].delay = 1000;
pmic.core.3.block[0].count = 1;
pmic.core.3.block[0].commands[0].0x07.0xFF = 0x2E;
# 2. Set GPIO3 Power down slot to 6.
pmic.core.3.block[1].type = 1; # I2C Type
pmic.core.3.block[1].i2c-controller-id = 4;
pmic.core.3.block[1].slave-add = 0x78; # 7Bit:0x3c
pmic.core.3.block[1].reg-data-size = 8;
pmic.core.3.block[1].reg-add-size = 8;
pmic.core.3.block[1].delay = 10;

```

10.9.6 AO Block Parameters

Parameter	Description
aoblock-count	Specifies the number of AO sets described in the file.
command-retries-count	Specifies the number of allowed command attempts.

Parameter	Description
delay-between-commands-us	<p>Specifies the wait timeout, in microseconds, before issuing the bus clear command. The wait time is calculated as:</p> $1 \ll n \text{ microseconds}$ <p>Where n is provided by this parameter.</p>
wait-before-start-bus-clear-us	Specifies the wait time, in microseconds, before issuing the bus clear command.
block-count	Specifies the number of blocks in the AO block.

10.9.6.1 I2C type-specific parameters

The I2C type specific parameters are as follows:

Parameter	Description
i2c-controller-id	Specifies the controller ID of I2C.
slave-add	Specifies the 7-bit secondary address.
reg-data-size	<p>Specifies the register size in bits:</p> <p>0 or 8:1 byte</p> <p>16: 2 byte</p>
reg-add-size	<p>Specifies the register address size in bits:</p> <p>0 or 8:1 byte</p> <p>16: 2 byte</p>

- Commands can be either MMIO or I2C.
- The information is in the format <address> = <data>, to support the write-only sequence.
- All commands are indexed, to facilitate multiple commands in a given block.
- Commands are sent to the device in sequence, starting from index 0, in the following format

The command syntax is as follows:

```
commands[command-index].<addr> = <data>;
```

All reset conditions support 3 AO blocks, initialized as follows:

```
bootrom.<reset-name>.aocommand[0] = <ao block ID>
bootrom.<reset-name>.aocommand[1] = <ao block ID>
bootrom.<reset-name>.aocommand[2] = <ao block ID>
```

Where <reset_name> is one of the following: watchdog5, watchdog4, sc7, sc8, soft-reset, sensor-aotag, vfsensor, or hsm.

For example:

```
bootrom.aoblock-count = 2;
# Automatic power cycling: Set MAX77620
# Register ONOFFCFG2, bit SFT_RST_WK = 1 (default is "0" after cold boot),
# Register ONOFFCFG1, bit SFT_RST = 1
bootrom.aoblock[0].command-retries-count = 1;
bootrom.aoblock[0].delay-between-commands-us = 1;
bootrom.aoblock[0].wait-before-start-bus-clear-us = 1;
bootrom.aoblock[0].block-count = 1;
bootrom.aoblock[0].block[0].type = 0; # I2C Type
bootrom.aoblock[0].block[0].slave-add = 0x3c; # 7Bit:0x3c
bootrom.aoblock[0].block[0].reg-data-size = 8;
bootrom.aoblock[0].block[0].reg-add-size = 8;
bootrom.aoblock[0].block[0].count = 2;
bootrom.aoblock[0].block[0].commands[0].0x42 = 0xda;
bootrom.aoblock[0].block[0].commands[1].0x41 = 0xf8;
# Shutdown: Set MAX77620
# Register ONOFFCFG2, bit SFT_RST_WK = 0
# Register ONOFFCFG1, bit SFT_RST = 1
bootrom.aoblock[1].command-retries-count = 1;
bootrom.aoblock[1].delay-between-commands-us = 1;
bootrom.aoblock[1].wait-before-start-bus-clear-us = 1;
bootrom.aoblock[1].block-count = 1;
bootrom.aoblock[1].block[0].type = 0; # I2C Type
bootrom.aoblock[1].block[0].slave-add = 0x3c; # 7Bit:0x3c
bootrom.aoblock[1].block[0].reg-data-size = 8;
bootrom.aoblock[1].block[0].reg-add-size = 8;
bootrom.aoblock[1].block[0].count = 2;
bootrom.aoblock[1].block[0].commands[0].0x42 = 0x5a;
bootrom.aoblock[1].block[0].commands[1].0x41 = 0xf8;
# Shutdown in sensor/ao-tag
#reset in soft reset.
# no commands for other case
bootrom.sensor-aotag.aocommand[0] = 1;
bootrom.soft-reset.aocommand[0] = 0;
```

10.9.7 Configuring the Security Configuration Registers

Tegra has separate registers for configuring bridge client security and bridge firewalls, known as security configuration registers (SCRs). SCRs are either configured by the

platform or re-configured for custom platforms. The custom configuration is provided using MB2 BCT at the MB2 stage. The SRC configuration file is available at:

```
<top>/drive-foundation/platform-config/platform/t23x/common/bct/firewall/tegra234-
firewall-config-base.dtsi
```

Format of the entries in dtsi:

```
reg@XYZ
{
    exclusion-info = <2>;           value = <0x80000000>; }
```

Exclusion-info is a bit-map (4 bits) with each bit signifying the following:

```
BIT[0] - SC7 SKIP
BIT[1:2] - PROGRAM_IN
  0 -> BEFORE_MB2
  1 -> MB2
  2 -> AFTER_MB2
BIT[3] - PRODUCTION ONLY
```

MB2 will program the SCRs only if the PROGRAM_IN field is set to MB2. SCRs that are to be programmed only on platforms with production fuses blown have PRODUCTION_ONLY flag set in exclusion-info. MB2 will check if the fuses are blown and programs the SCRs.

10.9.7.1 Usage

The command line usage syntax is as follows:

```
scr.<reg_index>.<exclusion-info> = <32 bit value>; # <reg_name>
```

Where:

- scr is the domain name prefix for the setting.
- <reg_index> is the matching MB1 and CFG file sequence, beginning at 0.
- <exclusion-info> is one of the values as follows:

Value	Description
0	Include: regular SCRs loaded from BCT in cold boot, from stored context in warm boot.
1	Exclude: Present data in the CFG file but do not load data from the BCT. Allows SCR programming in MB2 or later.
2	SC7 resume: Program from BCT in cold boot, but exclude for warm boot.

MB1 code lists SCR register absolute addresses in an indexed list.

For example:

```
# SCR register configurations
scr.134.5 = 0x3f008080; # APS_AST_SCR_AST_REG_3_SEC_CONTROL_0
scr.135.5 = 0x3f008080; # APS_AST_SCR_AST_REG_4_SEC_CONTROL_0
```

10.9.8 Miscellaneous Configurations

The miscellaneous configuration file is available at:

```
<top>/drive-foundation/platform-config/
bct/t194/misc/tegra194-mb1-bct-misc-<auto>.cfg
```

The fields contained in `misc<auto>.cfg` are as follows:

Field	Description	Configuration Example
enable_can_boot	Controls early CAN initialization. If set, spe-can firmware loading spe-r5 processor boot is done.	enable_can_boot = 1;
enable_blacklisting	Controls DRAM ECC blacklisting: 0: Disable ECC denylisting 1: Enable ECC denylisting	enable_blacklisting = 0;
disable_sc7	Controls SC7 state entry: 0: Enable sc7 1: Disable sc7	disable_sc7 = 0;
fuse_visibility	Certain fuses cannot be read or written by default because they are not visible. If this field is set, MB1 enables fuse visibility for such fuses.	fuse_visibility = 1;
enable_vpr_resize	Controls enablement of VPR resize functionality.	enable_vpr_resize=0

10.9.8.1 Debug

There are debug functionality which can be enabled or disabled using BCT flag.

Debug Control Fields	Description	Configuration Example
uart_instance	Configures the UART instance for console prints.	debug uart_instance = 1;
enable log	Enables/disables console logging.	debug.enable_log = 1;
enable_secure_settings	Unused.	-

10.9.8.2 AOTAG

The AO-TAG register is programmed in the MB1, which controls the maximum temperature at which Tegra platform is allowed to operate. If the temperature exceeds that limit, an automatic shutdown is triggered.

AOTag Control Fields	Description	Configuration Example
boot_temp_threshold	Boot temperature threshold in millicentigrade. If temperature is higher than the temperature specified in this field, MB1 waits or shuts down the device.	aotag.boot_temp_threshold = 105000;
cooldown_temp_threshold	Cool down temperature threshold in millicentigrade. MB1 resumes booting when the device has cooled to this threshold temperature.	aotag.cooldown_temp_threshold = 85000;
enable_shutdown	If set to 1, enables shutdown using aotag if temperature is above boot temperature threshold.	aotag.enable_shutdown = 1;

The clock control fields in the following table hold the clock divider values for the various modules that MB1 programs.

Clock Control Fields	Description	Configuration Example
bpmp_cpu_nic_divider	Program the cpu nic divider to control the BPMP CPU frequency. A value 1 less than the value in the field is directly written to the register.	clock_bpmp_cpu_nic_divider = 1;

Clock Control Fields	Description	Configuration Example
bpmp_apb_divider	<p>Program the apb divider to control the APB bus frequency.</p> <p>A value 1 less than the value in the field is directly written to the register.</p>	clock.bpmp_apb_divider = 1;
axi_cbb_divider	<p>Program the axi_cbb divider to control the AXI-CBB bus frequency.</p> <p>A value 1 less than the value in the field is directly written to the register.</p>	clock.axi_cbb_divider = 1;
se_divider	<p>Program the se divider to control the SE Controller frequency.</p> <p>A value 1 less than the value in the field is directly written to the register.</p>	clock.se_divider = 1;
aon_cpu_nic_divider	<p>Program the cpu_nic divider to control the AON(SPE) CPU frequency.</p> <p>A value 1 less than the value in the field is directly written to the register.</p>	clock.aon_cpu_nic_divider = 1;
aon_apb_divider	<p>Program the apb divider to control the AON(SPE) APB frequency.</p> <p>A value 1 less than the value in the field is directly written to the register.</p>	clock.aon_apb_divider = 1;
aon_can0_divider	<p>Program the can0 divider to control the CAN0 controller frequency.</p> <p>A value 1 less than the value in the field is directly written to the register.</p>	clock.aon_can0_divider = 1;
aon_can1_divider	<p>Program the can1 divider to control the CAN1 controller frequency.</p> <p>A value 1 less than the value in the field is directly written to the register.</p>	clock.aon_can1_divider = 1;
osc_drive_strength	Unused	-

Clock Control Fields	Description	Configuration Example
pllaon_divp	Program the P value of PLL-AON. A value 1 less than the value in the field is directly written to the register.	clock.pllaon_divp = 2;
pllaon_divn	Program the N value of PLL-AON. A value 1 less than the value in the field is directly written to the register.	clock.pllaon_divn = 25;
pllaon_divm	Program the M value of PLL-AON. A value 1 less than the value in the field is directly written to the register.	clock.pllaon_divm = 1;

10.9.8.3 AST setting

The AST settings for various firmware is loaded by MB1/MB2. These are the virtual addresses of the firmware. MB1/MB2 programs corresponding physical addresses based on the location where it loaded the firmware in memory (DRAM). Normally, it is not necessary to change these settings.

Fields	Description	Configuration Example
bpmp_fw_va	Virtual address for BPMP-FW	ast.bpmp_fw_va = 0x50000000;
mb2_va	Virtual address for MB2-FW	ast.mb2_va = 0x52000000;
sce_fw_va	Virtual address for SCE-FW	ast.sce_fw_va = 0x70000000;
apr_va	Virtual address for Audio-protected region used by APE-FW	ast.apr_va = 0xC0000000;
ape_fw_va	Virtual address for APE-FW	ast.ape_fw_va = 0x80000000;

10.9.8.4 I2C setting

The I2C settings specify the operating frequency of the I2C bus in MB1/MB2. The default is 100 KHz.

Field	Description	Configuration Example
0	Specify the clock for I2C controller instance 0	i2c.0 = 400;
4	Specify the clock for I2C controller instance 4	i2c.4 = 1000;

10.9.8.5 SW Carveout

The SW carveout settings specify the address and size for BL carveout.

Field	Description	Configuration Example
cpubl_carveout_addr	Start location of the CPU-BL Carveout	sw_carveout.cpubl_carveout_addr = 0x96000000;
cpubl_carveout_size	Size of the CPU-BL Carveout	sw_carveout.cpubl_carveout_size = 0x02000000;
mb2_carveout_size	Size of the MB2 Carveout	sw_carveout.mb2_carveout_size = 0x00400000;

10.9.8.6 CPU Param

The CPU parameter settings contain the initial settings passed to CPU-Init FW. Contact NVIDIA before changing these settings.

Field	Description	Configuration Example
Bootcpu	Specify Boot CPU. 4 means A57 cpu0 and 0 mean Denver0. For automotive applications use A57-cpu0.	cpu.bootcpu = 4
ccplex_platform_feature	Platform feature passed to the CPU-Init FW.	cpu.ccplex_platform_features = 0x581;
lsr_dvcomp_params_b_cluster	Contains setting for initializing ADC and DVC, which need to be functional before CPU rails are brought up	cpu.lsr_dvcomp_params_b_cluster = 0xC0780F05C;

Field	Description	Configuration Example
lsr_dvcomp_params_m_	Contains setting for initializing ADC and DVC, which need to be functional before CPU rails are brought up	cpu.lsr_dvcomp_params_m_cluster = 0xC0780F05C;
nafll_m_cluster_data	Initial NAFLL settings for cluster for Denver	cpu.nafll_m_cluster_data = 0x11F04461;
nafll_b_cluster_data	Initial NAFLL settings for cluster for A57	cpu.nafll_b_cluster_data = 0x11F04461;

10.9.8.7 Dev-param

The DEV parameters are the device settings used by MB1/MB2.

Field	Description	Configuration Example
qspi.clk_src	Specify the clock source. The value corresponds to what is mentioned in the QSPI CLK SRC register. 0: pllp_out0 4: pllc4_muxed	devinfo.qspi.clk_src = 0; #
qspi.clk_div	clk_div = N+1; Hence N = 3 & clk_rate = 163.2 MHz = (408 MHz / ((N / 2) + 1))	devinfo.qspi.clk_div = 4;
qspi.width	Specify the width of the QSPI BUS during transfer 0 : 1 bit (x1 mode) 1 : 2 bit (x2 mode) 2 : 4 bit (x4 mode)	devinfo.qspi.width = 2
qspi.dma_type	Specify which DMA to use for transfer if mode of transfer is DMA. For QSPI, in MB1/MB2, BPMP-DMA should be used. 0 : GPC-DMA 1 : BPMP-DMA	devinfo.qspi.dma_type = 1

Field	Description	Configuration Example
qspi.xfer_mode	Specify mode of transfer 0: PIO 1: DMA	devinfo.qspi.xfer_mode = 1;
qspi.read_dummy_cycles	The dummy cycles allow the device internal circuits additional time for accessing the initial address location. During the dummy cycles the data value on IOs are "don't care" and may be high impedance.	devinfo.qspi.read_dummy_cycles = 9
qspi.trimmer_val1	tx_clk_tap_delay for QSPI	devinfo.qspi.trimmer_val1 = 0
qspi.trimmer_val2	rx_clk_tap_delay for QSPI	devinfo.qspi.trimmer_val2 = 0

10.9.9 Enabling Memory Latent Fault Test

A memory latent fault test is used to test LPDDR memory. The purpose of the test is to detect if any memory faults are present in LPDDR. The test uses a specific data pattern write, read, and verify operations to check for memory faults. On detection of any faulty addresses, the pages are retired by adding them to a list and not used for allocation. The test is done at Key-On/Key-Off IST and covers the entire memory.



Note: Enabling latent fault test will increase the IST timeout in NVIDIA DRIVE® OS MCU software.

The following example shows how to enable this feature:

```
diff --git a/drive_av/qnx/platform_config_profile.json b/drive_av/qnx/platform_config_profile.json
index e8f4a40..56aa5c3 100644
--- a/drive_av/qnx/platform_config_profile.json
+++ b/drive_av/qnx/platform_config_profile.json
@@ -110,6 +110,9 @@
#endif
#ifndef ENABLE_PVA
"ENABLE_PVA",
#endif
#ifndef ENABLE_LFT
+ "ENABLE_LFT",
#endif
"ENABLE_HV_LOAD",
"ENABLE_DISP_LA_PTSA",
@@ -147,6 +150,9 @@
#endif
#ifndef ENABLE_PVA
```

```
"ENABLE_PVA",
+#endif
+##ifdef ENABLE_LFT
+ "ENABLE_LFT",
#endif
"ENABLE_HV_LOAD",
"ENABLE_DISP_LA_PTSA",
@@ -158,6 +164,9 @@
#endif
#ifndef ENABLE_PVIT
"ENABLE_PVIT",
+##endif
+##ifdef ENABLE_LFT
+ "ENABLE_LFT",
#endif
"ENABLE_HV_LOAD",
"ENABLE_CCPLEX_SMMU_PTW"
```

10.10 Restricting Power Controls

Because maintaining a safe operating state is crucial, the power controls, including clocks, resets, and power states of NVIDIA Tegra® hardware IPs, are access restricted through firewalls to the BPMP-FW software.

Chapter 11. Mass Storage Partition Configuration

This section describes how to configure the mass storage partition.

The platform supports formatting mass storage media into multiple partitions for storing data, such as device OS images and boot loader images. Data inside these partitions are not end-user visible through the typical OS filesystems.

The platform also supports the GUID partition table GPT scheme for defining the layout of the partition table on a physical hard disk.

Creating Partitions

Options for creating partitions differ between boot media and non-boot-media.

- For boot media (for QSPI media), the partition layout is passed from the kernel command line or via the DT node. The partition layout is a kernel interpretation. The boot loader continues to use the partition table (PT).
- For non-boot-media (eMMC/SD card), you can create partitions from user-space or by using bootburn. However, if you update the partition from user-space, the partition layout for the boot loader will go out of sync with the partitions. This is because user-space uses GPT for storing partition layout. In contrast, Boot loader uses the PT definitions.

For best practices, expose **non-system partitions** to user-space, excluding the **system partitions** like BCT, PT, boot loaders, and kernel partitions. To update system partitions using update tools, it is recommended that whole flash be exposed to user-space in the kernel.

For information about the flashing flow, see [Flashing with Bootburn](#).

Multipartition Architecture

The partition properties are:

- Base address, expressed as bytes
- Length, expressed as bytes
- Name, maximum 20 character-length string
- Storage device ID
- File system type
- Flag indicating whether the partition is to be write-protected on boot.

11.1 Partition Overview

Two configuration (CFG) versions are supported. The existing CFG format is identified as “legacy” or “v1,” while the 3-level CFG format is identified as “v2.”

The usage for each type of partition is as follows:

- [Guest OS Partitions](#)
- [Native Partitions](#)

11.1.1 Native Partitions

The partition names for the primary and recovery partitions are as follows.

- For CFG v1, each recovery partition’s name is the primary partition’s name suffixed with ‘r’.
- For CFG v2, only the recovery partitions in the top-level CFG have their names suffixed with ‘r’.
- The other recovery partitions in CFG v2 maintain the same name as their primary counterparts; the recovery partitions are differentiated based on the boot chain they belong to (i.e. A or B).

Partition name	Applies to	Description
Bct		Contains the Boot Configuration Table.
mb1-bootloader		Contains the MB1 boot loader.
mb1-bct		Contains the BCT configuration file for MB1.
mb2-bootloader		Contains the MB2 boot loader for the BPMP.
mts-preboot		Contains preboot firmware for the CPU.
mts-bootpack		Contains the CPU firmware.
bpmp-fw		Contains the power-management firmware.
bpmp-fw-dtb		Contains the power-management firmware DTB.
ramdisk		Contains the primary copy of the ramdisk image.

Partition name	Applies to	Description
secure-os		Contains the recovery copy of the ramdisk image.
Pt		Contains the Partition Table.
kernel		Contains the kernel image.
kernel-dtb		Contains the device tree binary image required by kernel.
eks		Contains the eks.dat object, which is an encrypted Widevine key.
spe-fw		Contain the firmware for SPE-R5.
sc7-fw		Contains the firmware for SC7 resume.
sce-fw		Contains the firmware for SCE-R5. This can be either Camera firmware or Safety firmware, based on use case.
adsp-fw		Contains the firmware for Audio processor.

Partition name	Applies to	Description
gp1 or fs-gp1		<p>Contains the primary GUID Partition Table.</p> <p>The GPT partitioning scheme must be used. Extended boot record partitions (ER<n>) are not required with GPT partitioning. Do not use the filename partition attribute when creating a GP1 partition.</p> <p>This partition type signifies the start of the GPT partitioning. All the partitions between GP1 to GPT partition are counted as partitions in GPT creation.</p> <p>GP1 partition size must be greater than or equal to 17,408 bytes. Similarly, GPT partition size must be greater than or equal to 16,896 bytes.</p> <p>For more information about using GPT, see Configuring GPT Devices.</p>
gpt or fs-gpt		<p>Contains the secondary GUID Partition Table. This partition must be at the end of the device.</p> <p>The GPT partitioning scheme must be used. Extended boot record partitions (ER<n>) are not required with GPT partitioning. Do not use the filename partition attribute when creating a GPT partition.</p> <p>GPT partition size must be greater than or equal to 16,896 bytes.</p> <p>For information about using GPT, see Configuring GPT Devices.</p>
A_<type>_chain or B_<type>_chain <small>NVIDIA DRIVE OS Linux SDK Developer Guide</small>		<p>Contains the partitions that belong to a particular boot chain (i.e. A or B). It points to a subsidiary CFG file that defines those partitions. <type> indicates the storage device type as emmc or qspi.</p>

11.1.2 Guest OS Partitions

The supported guest partition names, and their legal values, is as follows.

Partition Name	Description
Applies to DRIVE OS Linux: guest-linux	In CFG v1, specifies the global partition for the first guest. In CFG v2, specifies the global partition for any given guest. In either case it points to sub_storage_cfg, which defines the internal layout of that guest.
gos0-gp1	Specifies the primary GPT partition for the storage device used by Guest OS 0 (GOS0).
gos1-gp1	Specifies the primary GPT partition for the storage device used by Guest OS 1 (GOS1).
gos0-gpt	Specifies the back-up GPT partition for the storage device used by GOS0.
pt	Specifies the NVIDIA-specific implementation of the Partition Table, which stores the information on the locations of the partitions in storage_cfg.

11.2 Customizing the Configuration File

Customize the mass storage partitions on the target by modifying the configuration file flashed to the target.

Configuration files (CFG files) consist of mass storage device declarations followed by partition declarations. The NVIDIA tools use the configuration file to create the images on the host and flash the target device on the specified memory locations.

The platform includes a default configuration file used by the flashing scripts. Use the default configuration file as a starting point for any customization, and backup the original file before attempting modifications.

For an example of the dual Linux OS partition configuration, see the Example Virtual Partition Configuration chapter in the *NVIDIA DRIVE OS 6.0 Linux PDK Development Guide*. For an example of a native OS (not virtual) partition configuration, see the Example Native OS Partition Configuration chapter in the *NVIDIA DRIVE OS 6.0 Linux PDK Development Guide*.



Note:

Two CFG versions are supported. The existing CFG format is identified as “legacy” or “v1,” while the 3-level CFG format is identified as “v2.”

The default configuration file for a platform is adequate for initial product development. However, consider creating a custom configuration file for these stages:

- > Finalizing production
- > Flashing updated images

There are several advantages to using a custom configuration file. It can specify:

- > Different kernel images for recovery and primary partitions
- > Temporary partition
- > RamDisk or other feature in recovery partition or vice-versa

If there is a need to repeatedly switch between NFS and MMC, it is recommended to create and use separate configuration files for each.

11.2.1 Customizing Partitions

When partitions are finally customized (for example, you have modified or added partition definitions in the configuration file, run the flashing script to flash the device and set the partitions as defined.

By default, the flashing scripts use the default configuration file. If you customized a configuration file, specify that file when calling the flash script.

For more information, consult [Flashing the Board](#).

11.2.2 Configuration File Entries

Declarations consist of attribute/value pairs. The configuration file format is:

- > [meta]—(optional) Specifies CFG metadata.
- > <attr>=<val>—Specifies attribute/value pairs for the device.
- > [device n]—(required) Specifies a mass-storage device.
- > <attr>=<val>—Specifies attribute/value pairs for the device.
- > [partition]—Specifies a partition in the current device.
- > <attr>=<val>—Specifies attribute/value pairs for the partition.
- > # - comments

Mandatory configuration file entries are:

- > Device type.
- > Partition for the device partition table.
- > Partition for the OS image.
- > Partition for the kernel.
- > Partition for the boot loader.
- > BCT partition for the boot configuration table.
- > Size of partition, which can be the exact file size being programmed in bytes provided it be `erase_block_size` aligned for the flash. However, to minimize changes to partition

sizes in the CFGpartition, it is recommended to set the partition size in the CFG file greater than actual file size to be flashed in the partition.


Note:

The partition size always aligns to the erase block size.

11.2.3 Setting Attributes

The possible values for the meta, device, and partition attributes are as follows.

11.2.3.1 Meta Attributes Table

The following table shows supported device attributes and their legal values.

Device	Values	Description
Version	1 (default) or 2	<p>Specifies the CFG version.</p> <ul style="list-style-type: none"> • 1: legacy (v1) • 2: 3-level (v2)

11.2.3.2 Device Attributes Table

The following table shows supported device attributes and their legal values.

Device	Values	Description
type	qspi/spi sdmmc ide ufs_lun ufs_boot	Specifies the type of mass storage device.

Device	Values	Description
instance	<instance_num>	<p>Specifies the controller instance.</p> <p>Xavier has four sdmmc controller instances. Each instance drives one end point, as follows:</p> <ul style="list-style-type: none"> • instance=0 specifies SDMMC1. • instance=1 specifies SDMMC2. • instance=2 specifies SDMMC3. • instance=3 specifies SDMMC4. <p>For type-ufs, the instance will correspond to the SCSI device instance. System partitions, if any, must be present on LUN 0, and LUN 0 must enumerate at /dev/sda.</p>

Device	Values	Description
size	<device_size>	Specifies the size of the mass storage device. Required if GPT partition is created during bootburn.
linux_name	<linux_name>	<p>Specifies the name of the device in bootburn.</p> <p>For QSPI devices: /dev/block/mtdblock<id> Where <id> identifies the device.</p> <ul style="list-style-type: none"> For MMC devices: /dev/block/mmcbblk<id> For mSATA/USB/UFS devices: /dev/block/sda <p>Note: With multiple mSATA/USB devices connected, enumeration varies as sda, sdb, etc. Instead of path, <linux_name> is specified with the controller address-space. For example: /dev/block/34600000.sdhci</p>
lun	0-7	Logical unit number for the device.

11.2.3.3 Partition Attributes Table

The following table shows supported partition attributes and their legal values.

Partition Attribute	Values	Description
encryption_derivation_string	Binary Hex String	For normal partitions, this is a max 32 byte hex string. For name=bct, this is a max 8 byte hex string. The string can be proceeded by 0x. Default is zero.

Partition Attribute	Values	Description
derivation_const1_string	Binary Hex String	For name=bct only, this is a max 16 byte hex string. The string can be proceeded by 0x. Default is zero.
derivation_const2_string	Binary Hex String	For name=bct only, this is a max 16 byte hex string. The string can be proceeded by 0x. Default is zero.
derivation_const_tz_string	Binary Hex String	For name=bct only, this is a max 16 byte hex string. The string can be proceeded by 0x. Default is zero.
derivation_const_gp_string	Binary Hex String	For name=bct only, this is a max 16 byte hex string. The string can be proceeded by 0x. Default is zero.
derivation_const_fsi_string	Binary Hex String	For name=bct only, this is a max 16 byte hex string. The string can be proceeded by 0x. Default is zero.

Partition Attribute	Values		Description
	Bits	Description	
virtual_storage_ivc_ch	32-bit unsigned number. The following table describes the virtual_storage_ivc_ch attribute bit field:		Every virtualized partition must have the virtual_storage_ivc_ch attribute. The VSC server relies on this attribute to configure the virtualization aspects of storage partitions.
	31	Is Virtual Storage Flag [virt = 1, non-virt = 0]	
	30-24	Storage Server ID [int value from 0-0x7F]	
	23	Shared Partition Flag [Shared = 1 , Exclusive = 0]	
	22	Reserved for attributes.	
	21:18	Partition Priority. [Values from 1 thru 5 where 1 is highest priority]	
	17	Disable Pass-through [1 = Disable, 0 = Enable]	
	16	Read only flag [RO = 1, RW = 0]	
	15:8	Mempool ID [int value from 0-0xFF]	
	7:0	IVC Queue	

Partition Attribute	Values	Description
encryption	<true or false>	Optional: true to encrypt with either global key or encryption_key.
encryption_key	<filename>	Optional: Partition-specific encryption key. Encryption must be true for partition to be encrypted.
name	<name>	Specifies up to a 20-character name for the partition. This name is used when opening a partition for read/write access. Names longer than 20 characters are not supported.
id	<identifier>	Identifier.
type	boot_config_table bootloader partition_table extended_boot_record GP1 GPT data	Specifies the type of partition. <ul style="list-style-type: none">• boot_config_table is for the BCT.• partition_table is for the partition table.• extended_boot_record is for the EBR.• GP1 is for the primary GPT partition.• GPT is for the secondary GPT partition.• data is for the remaining partitions.

Partition Attribute	Values	Description
type (continued)	mb1_boot_config_table mb2_bootloader mts_preboot mts_bootpack bpmp_fw bpmp_fw_dtb bootloader_cpu secure_os kernel_dtb kernel ramdisk WBO spe-fw ape-fw sce-fw	
allocation_policy	absolute sequential extended	<p>Specifies the type of allocation policy.</p> <p>sequential—The begin immediately after the preceding partition.</p> <p>absolute—The partition begins at the location specified by the start_location attribute.</p> <p>extended—The partition is extended until end of the device just before GPT partition. Only one extended partition is allowed per device.</p>

Partition Attribute	Values	Description
filesystem_type	basic ext2 ext3 ext4 qnx	<p>Specifies the type of filesystem formatted on the partition when the partition is created. The value may be:</p> <p>basic—No file system. The partition can be overloaded with a file system image by providing a <i>filename</i> for the partition (see the <i>filename</i> entry in this table), or by formatting the partition after it is created.</p> <p>ext2, ext3, or ext4—An ext2, ext3, or ext4 filesystem.</p> <p>Applies to: eMMC, SD card, and NOR media in Linux; UBIFS for QSPI media.</p> <p>qnx—A qnx6 filesystem.</p> <p>Applies to DRIVE OS QNX:</p> <p>External file system support depends on the support available in the flashing tool.</p> <p>For a qnx6 filesystem the following environment variables must be set:</p> <pre>\$ export QNX_HOST=<QNX_TOOLCHAIN_BASE>/host/linux/x86_64/</pre> <pre>\$ export QNX_TARGET=<QNX_TOOLCHAIN_BASE>/target/qnx7/</pre> <p>Where <QNX_TOOLCHAIN_BASE> is the pathname of the toolchain base.</p>

Partition Attribute	Values	Description
start_location	<start>	Specifies the starting location of the partition in the mass storage device. Valid for absolute partitions only.
size	<size>	Specifies the size of the partition in bytes. Decimal and hexadecimal values are valid. The boot loader requires that the partition size be aligned and the alignment size be: <ul style="list-style-type: none"> • For QSPI: 128K (131072) bytes • For eMMC/SD card: 8K (8192) bytes

Partition Attribute	Values	Description
partition_attribute	32-bit unsigned number	<p>For CFG v1:</p> <p>For virtualization, this property specifies to which guest this partition belongs.</p> <p>Partition_attribute must be equal to guest_id + 1 as defined in the PCT. Additionally, per device only one partition can be allocated to a given guest.</p> <p>For CFG v2:</p> <p>The bits of partition_attribute have the following definitions:</p> <ul style="list-style-type: none"> • Bit 31: Set if the partition's storage device is the boot medium for the guest. • Bit 30: Set if the partition holds a blob for boot-chain (and sub_cfg_file is present if the partition is present on the global boot device). • Bit 29: Set if the partition holds a blob for guest partitions (and sub_cfg_file is present). • Bit 28: Set if the partition holds a blob for user partitions (and sub_cfg_file is present). • Bits 4:0: guest ID.
filename	<p><filename></p> <p>See Notes at the end of this table.</p>	<p>Specifies the name of the file to write into the partition. The file must be present in the directory in which nvimagegen is running or named with an absolute path.</p> <p>The filename attribute cannot be used with the following attributes:</p> <ul style="list-style-type: none"> • dirname • imagepath

Partition Attribute	Values	Description
rcm_filename	<p data-bbox="638 287 780 318"><filename*></p> <p data-bbox="638 375 959 439">See Notes at the end of this table.</p>	<p data-bbox="1041 287 1393 515">Specifies the recovery mode binary for use in microboot 1 (NVC) or microboot 2 (MB2). The file must be present in the directory in which nvimagegen is running or named with an absolute path.</p> <p data-bbox="1041 551 1383 646">The filename attribute cannot be used with the following attributes:</p> <ul data-bbox="1041 682 1155 713" style="list-style-type: none"> <li data-bbox="1041 682 1155 713">• dirname
dirname	<p data-bbox="638 766 845 798"><Directory Path*></p> <p data-bbox="638 855 959 918">See Notes at the end of this table.</p>	<p data-bbox="1041 766 1421 994">Specifies the directory path for creating the filesystem image (based on the filesystem_type attribute) and burning the same to the media. This attribute applies to the ext2/3 external file system.</p> <p data-bbox="1041 1030 1416 1094">For detailed information, see the *_fs.cfg file.</p> <p data-bbox="1041 1129 1411 1214">The dirname and filename attributes cannot be used in the same partition.</p>
imagepath	<p data-bbox="638 1273 784 1305"><file name*></p> <p data-bbox="638 1362 959 1425">See Notes at the end of this table.</p>	<p data-bbox="1041 1273 1413 1501">Specifies the kernel image path, which is usually an OS kernel partition. The Flashing tools create the kernel image, which will be flashed based on the other specified. For detailed information, see the *_fs.cfg file.</p> <p data-bbox="1041 1537 1411 1632">The imagepath and filename attributes cannot be used in the same partition.</p>

Partition Attribute	Values	Description
os_args	<String>	<p>Kernel command line to be passed to the kernel. Key-value pairs are separated by spaces. root= is required and, depending on the root=value, rootfstype= may also be required.</p> <p>The supported key-value pairs are specified in:</p> <p>kernel/Documentation/kernel-parameters.txt</p> <p>For more information, see Aligning os_args and the Mass Storage Layout.</p> <p>Valid only with the kernel_dtb partition.</p>
ramdisk_path	<filename*> See Notes at the end of this table.	<p>Specifies the name of the ramdisk image file. The specified file must be present in the directory in which nvimagegen is running or the name must be an absolute path.</p> <p>Valid only with the imagepath attribute.</p>
os_load_address	<address>	<p>Specifies address where boot loader must load the kernel image.</p> <p>When the image is zImage (specified in imagepath), set this property to 0xA00800. When the image is Image, set it to 0x8000.</p>
ramdisk_load_address	<address>	Specifies the load address for Ram-disk.

Partition Attribute	Values	Description
decompression_algorithm	<lzf zlib none>	<p>Specifies the algorithm that Quickboot uses for decompressing images.</p> <p>lzf is the preferred algorithm.</p> <p>Ensure that the partition has enough space to hold the image, especially when flashing an uncompressed image (decompression_algorithm=none).</p> <ul style="list-style-type: none"> If this field is not specified, by default, lzf is used as the decompression algorithm. Quickboot decompresses images in parallel with loading the image. Use only when the imagepath or ramdisk_path attribute points to an image.
Applies to: QSPI media only. write_protect	0 or 1	<p>Not currently supported.</p> <p>Specifies whether to write-protect a given partition. Write-protection prevents partition erasures.</p> <p>This attribute has no effect during flashing. All partitions are unprotected before flashing.</p>
sub_config_file	<cfg.pathname*> See Notes at the end of this table.	<p>Specifies the path to the configuration file that contains the layout for the guest. The flashing system, including the main configuration file, assumes this is a single blob file.</p>
load_address	Numeric_value	<p>Specifies the load address for the binary in the given partition.</p>
entry_point	Numeric_value	<p>Specifies the entry_point for the binary in the given partition.</p>

Partition Attribute	Values	Description
version	Numeric_value	Specifies the version for the given binary.
image_type	<linux android integrity hypervisor ramdisk mods>	Specifies the type of image. Valid only for kernel and ramdisk partitions. Currently, mods is for internal use.
stream_validation	<yes no>	Specifies whether to validate the image in parallel with loading and decompressing the image. If decompression_algorithm is selected, this attribute is ignored.
authentication_group	<number>	Specifies the partition to a group. All the partitions grouped will be authenticated together. The minimum group number is 1 and the maximum group number is 15. Group number 0 means not in a group. For more information, see Grouping of Boot Images.
virtual_storage_ivc_ch	32-bit unsigned number	Every virtualized partition must have the virtual_storage_ivc_ch attribute. The VSC server relies on this attribute to configure the virtualization aspects of storage partitions.
ispersistent	<yes no>	If ispersistent=yes, an image file is specified for the partition and --init-persistent-partitions is specified on bootburn.py or create_bsp_images.py command line and the value specified in the image file is written to persistent partition.

The following table describes the virtual_storage_ivc_ch attribute bit field.

Bits	Description
31	Is Virtual Storage Flag [virt = 1, non-virt = 0]
30-24	Storage Server ID [int value from 0-0x7F]
23	Shared Partition Flag [Shared = 1 , Exclusive = 0]
22	Reserved for attributes.
21:18	Partition Priority. [Values from 1 thru 5 where 1 is highest priority]
17	Disable Pass-through [1 = Disable, 0 = Enable]
16	Read only flag [RO = 1, RW = 0]
15:8	Mempool ID [int value from 0-0xFF]
7:0	IVC Queue ID [int value from 0-0xFF]

Example: Virtualized GOS persistent storage with IVC Queue=0xEC, Mempool ID=0x0x35, Read Write partition, Passthrough disabled, Priority Partition of 4, exclusive partition.

```
[partition]
name=gos0-misc-pers
allocation_policy=sequential
filesystem_type=basic
size=0x6600000
partition_attribute=<GID_GUEST0_VM+1>
virtual_storage_ivc_ch=0x8<GID_VSC_SERVER>1235EC
```



Note: *To avoid Bootburn errors, do not use special characters in directory names. Such special characters include the plus sign (+) and pound sign (#).

Configuration files that configure the Flashing tool for an eMMC, or QSPI media depend upon the targeted OS. You can find the configuration files in these locations:

```
<top>/drive-foundation/tools/flashtools/bootburn_t23x/*.cfg
```

For best practices, retain the file as provided and add optional partitions based on project requirements.

11.2.4 Aligning os_args Values and the Mass Storage Layout

When flashing a device, the flashing script uses the configuration file to determine how to flash individual partitions. `os_args` attributes provide a kernel command line that specifies file system and partition attributes. Of particular interest are the settings for Memory Technology Device (MTD) devices, which use QSPI flash.

If you change your flash partition layout, you must also modify the attributes in `os_args` to be consistent with the mass storage layout. For example, if you move the data partition, you must also update the offsets or indices in the `os_args` value. Such modifications are required to ensure non-overlapping partitions.

By default, the OS kernel has no knowledge of the mass storage layout created when flashing (i.e., QSPI partition offset and size, and file system type). At runtime, the kernel cannot determine all layout settings. In particular, the `os_args` attribute may need to specify the Linux kernel settings described in the following table.

<code>os_args</code> Settings	Description
<code>rootfstype=ubifs</code>	Required if the root file system is UBIFS. Unlike the ext2/ext3/ext4 file systems, the kernel cannot at runtime detect UBIFS.
<code>root=ubi<n>_<m></code> Where: <n> is the sequential number of the UBI device (usually 0) <m> is the "id" attribute on the partition.	Required if the root device is a UBI volume. The alternative form of this value is <code>ubi<n>:<name></code> , where <name> is the partition name, such as <code>EARLY_FS_VIDEO</code> .
<code>ubi.mtd=<x></code> Where <x> is an MTD partition number.	Required to attach the UBI to the MTD partition at boot, for example, if the root device is a UBI volume.
<code>mtdparts=tegra-nor:<size>@<offset>(<name>)</code> Where: • <size> is the partition size in kilobytes. • <offset> specifies the beginning of the partition. • <name> is the symbolic name for the partition. For example: <code>whole_device</code> or <code>userspace</code> .	Required to export NOR partitions as mtdN and mtdblockN devices. For example, use this key-value pair to export the partition at offset <offset> so UBI can attach itself to such devices. You can get the value for <size> and <offset> from the default CFG file for your platform. <code>tegra-nor</code> is the flash device name and must not be changed.

For information on all supported kernel command parameters, see the following file.

`kernel/Documentation/kernel-parameters.txt`

11.2.5 Configuring GPT Devices

GUID Partition Table (GPT) is a standard for the layout of the partition table on a physical hard disk. For general information on GPT, see *GUID Partition Table* at:

http://en.wikipedia.org/wiki/GUID_Partition_Table

1. In the kernel configuration file, enable the following partition types:

```
CONFIG_PARTITION_ADVANCED=y
CONFIG_EFI_PARTITION=y
```

2. In your flash.cfg configuration file, specify the GP1 partition.

Partitions between this partition and GPT are exposed, although only those selected to mount are mounted.

Among other attributes for GP1 partitions, the following attributes must be set as specified here:

```
filesystem_type=basic
type=GP1
```

3. Specify the GPT partition.

Among other attributes for GPT partitions, the following attributes must be set:

```
filesystem_type=basic
type=GPT
size=0xFFFFFFFFFFFFFF
```

4.  **Note:** The GPT partition must be the last partition.
5. Specify fill and extension attributes, depending on your platform. For Orin, set the GPT partition `allocation_policy` attribute to `sequential`.

11.3 Flashing Partitions with a File System and Kernel Image

When the configuration file contains `imagepath` partition attributes, the flashing script flashes a file system and kernel image. Internally, flashing tools create the kernel image that is flashed based on the other partition attributes. The following partition attributes affect the file system:

- > `filesystem_type`
- > `filename`
- > `dirname`
- > `imagepath`

In addition to these attributes, the `imagepath` or `ramdisk_path` attribute uses the following attributes:

- > `os_args`
- > `ramdisk_path`
- > `os_load_address`
- > `stream_decompression`
- > `decompression_algorithm`
- > `os_load_address/ramdisk_load_address` or `load_address`

- > stream_validation
- > decompression_algorithm

See the Example Native OS Partition Configuration chapter in the *NVIDIA DRIVE OS 6.0 Linux PDK Development Guide* for an example that uses the imagepath and dirname partition attributes.



Note: Access to the *NVIDIA DRIVE OS Linux PDK Developer Guide* requires specific agreements with NVIDIA. Consult with your NVIDIA Customer Support Engineer for more information.

The flashing script invokes Flashing tools to perform actions for specific partitions.

For the dirname partition attribute:

- > Flashing tool creates the file system image based on filesystem_type partition attribute.
- > Bootburn then sends that image for writing to the media. This is the same as using the filename option in the configuration file.

The Flashing tools perform other actions.

For the imagepath and ramdisk_path partition attribute, the Flashing tool does the following:

- > Creates a compressed image based on the decompression_algorithm partition attributes.
- > Creates a final kernel image.

11.4 Managing Mass Storage Partitions in Virtualization

Virtualization enables managing independent partitions. It supports independent flashing, loading, and restarting of individual VM partitions.

You can enable this feature for individual partitions using the Partition Configuration Table (PCT) by setting a load_using_pl flag in your partition configuration.

If this flag is not enabled for a partition, then during the binding step all partition images are combined into a single bootable image with Hypervisor and other Foundation components. This image gets loaded into memory at once during system boot and reloading or rebooting such a partition at runtime is not possible.

Enabling the load_using_pl flag for a partition specifies that the system:

- > Stores partition images independently (boot loader, OS kernel, file system)
- > Enables independent flashing of partition images
- > Requires the Partition Loader to load and execute partition boot images
- > Enables partition restarting

11.4.1 Partition Loader

The Partition Loader is a special purpose boot loader image that is embedded into Hypervisor. During the boot process, Hypervisor:

1. Creates a VM container according to the PCT
2. Maps a Partition Loader image into guest OS memory space
3. Then passes control to it

Partition Loader is responsible for loading and starting guest OS boot images.



Note: While being a part of the Hypervisor image, Partition Loader is executed in a context of a guest OS VM and can only access physical memory or storage devices that belong to the OS according to the PCT.

When a partition restarts after resetting partition state, Hypervisor passes control back to Partition Loader for subsequent reloading of Guest OS images.

11.4.2 Storage Layout

Independent partition flashing and loading requires special storage device layout configuration. Configuration information is part of PCT and is used by flashing tools to program partition images and Partition Loader to load the images during system boot or partition restart.

For CF v1, the storage layout is defined by the following 2 configuration files:

- > Global layout configuration file—defines partitions that are globally visible on all storage devices in the system, such as boot loader, Foundation image, and root file system images.
- > VM partition layout configuration file—defines storage partitions that are visible to individual VMs. It describes the locations of partition images (OS loader, OS kernel) and boot parameters, such as OS command line.

For CFG v2, the storage layout is defined by these configuration files:

- > Global layout configuration file: Defines partitions that are globally visible on all storage devices in the system and are common to both boot chains, such as BR BCT and MB1 boot loader.
- > Boot chain layout configuration file: Defines partitions that are globally visible on all storage devices in the system and belong to a particular boot chain (i.e. A or B), such as Foundation image, and root file system images.
- > VM partition storage layout configuration file: Defines storage partitions that are visible to individual VMs. This file describes the locations of partition images (OS loader, OS kernel) and boot parameters, such as OS command line.

The above storage layouts are defined using the NVIDIA Mass Storage configuration format.

For more information, see [Partition Storage Layout Configuration Example](#).

11.4.2.1 To change the global layout

- > In the PCT directory, edit the following file:

```
<global_storage_qspi.cfg>
```

- > Where `<global_storage_qspi.cfg>` is the name of the file referenced by the `sub_config_file` property in the virtual machine (VM) partition layout file. There are no requirements for naming VM partition layout files.

11.4.2.2 Examples

The PCT examples are available at:

```
<top>/drive-foundation/virtualization/pct
```

The default configuration files for DRIVE OS Linux are as follows.

Global Storage Configuration File	Description
Dual Linux setup	/linux-linux/global_storage_qspi.cfg
First Linux partition storage configuration	/linux-linux/linux1_storage_emmc.cfg
Second Linux partition storage configuration	/linux-linux/linux2_storage_emmc.cfg

Chapter 12. NVIDIA DRIVE Utilities

Use the information in this section to understand the utilities available in this NVIDIA DRIVE® product. This information includes topics such as how to use PuTTY and Minicom utilities to communicate with the boards on your platform.

12.1 Device Tree Structure Include (DTSI)

A device tree is a tree-structured data format that represents information about the devices on a board.

Using device trees provides:

- Fewer "machine code" and "board" files
- A single unmodified kernel used for many platforms

For developing a product with the platform, the device tree data is automatically included in the flashed image.

The device tree data must be included in the flashed image when upgrading a product from an earlier release. Future kernel versions are expected to support device trees and to deprecate board files.

12.1.1 Device Tree Format

A device tree is a tree structure containing kernel-level information about hardware, including:

- Device characteristics
- Connections between devices (buses)
- Device configuration information

A device tree represents the hardware. The information in a device tree comes from sources such as hardware specifications and board schematics.

Device tree data is represented in the following formats:

- Device Tree source files (*.dts and *.dtsi files)
- Flattened device tree (FDT) structures

The resolved DTS and DTSI files for a device are represented as an FDT structure.

For information on the FDT format, see: http://elinux.org/Device_Trees.

- > Binary files known as device tree blobs or DTBs (*.dtb files)

The device tree compiler (DTC) compiles the FDT structure into a DTB file.

12.1.2 Device Tree Files in the BSP Framework

The BSP framework from NVIDIA includes device tree source files that describe the supported platforms. It provides source files that support hardware differences among the variants of each platform.

A hierarchy of DTSI files describes each platform. From highest (most general) to lowest (most specific), the hierarchy levels are:

A hierarchy of DTSI files describes each platform:

```
|-- Base Platform DTS
  |-- Platform Intermediate DTSI
    |-- Platform Common DTSI
      |-- SoC DTSI
```

Files named with this syntax contain a device tree for the platform with the specified Tegra version.

The SDK supplies the DTS files in the following directories:

```
<top>/hardware/nvidia/soc/t234/
<top>/hardware/nvidia/platform/t234
```

- > `tegra234-vcm31_<e3550a01-t186a | e3550a01-t186b | e3550a03-t186a | e3550a03-t186b>-base-* .dtsi`

Files named with this syntax contain chip-specific nodes common for boards that have the Tegra version in question.

- > `tegra234-vcm31t234-common .dtsi`

Files named with this syntax contain a Device Tree for the vcm31 platform with the specified Tegra version.

DTS and DTSI files refer to each other using `include` statements. This allows the reuse of pre-existing device tree nodes. A board-00 DTS file includes the relevant board DTSI file, which includes the relevant chip-specific DTSI file, which in turn includes the relevant SoC DTSI file.

12.1.3 Example: Platform Common DTSI

The `tegra234-soc/tegra234-common.dtsi` file is included in the platform intermediate DTSI file for each platform. It is included in the platform intermediate DTSI.

```
#include "address_map_new.h"
#include "clk-t234.h"
#include "reset-t234.h"
```

```
#include "tegra234-irq.h"
#include "tegra234-vcm31-thermal.dtsi"
#include "tegra234-camera.dtsi"
#include "tegra234-safety-sce.dtsi"
#include "t18x/tegra234-gpcdma-sid.h"
#include "t18x/tegra-t23x-agic.h"
#include "tegra234-soc-prod.dtsi"
{
    #address-cells = <2>;
    #size-cells = <2>;
    chosen {
        stdout-path = &uartb;
    };
}
```

12.1.4 Example: SoC DTSI

The `tegra234-soc/tegra234-common.dtsi` file is the SoC-specific DTSI file. It is included in the platform common DTSI.

```
#include "address_map_new.h"
#include "clk-t234.h"
#include "reset-t234.h"
#include "tegra234-irq.h"
#include "tegra234-camera.dtsi"
#include "t18x/tegra234-gpcdma-sid.h"
#include "t18x/tegra-t23x-agic.h"
{
    #address-cells = <2>;
    #size-cells = <2>;
    chosen {
    };
}
```

12.1.5 Viewing Pinmux Settings in the DTS

Quickboot configures the pinmux; the kernel does not configure them.

The device tree entries for pinmux are visible in the following `procfs` location. It contains the different states defined for dynamic pinmux configuration.

```
/proc/device-tree/pinmux@70000868
```

12.1.6 Kernel DTS Compilation and Flashing

The framework build system compiles the kernel DTS and DTSI files for your platform into DTB data, which is included in the flash image. Along with the kernel, DTB files are built automatically.

The boot loader uses the `libfdt` library to manage DTB data. With this library, Quickboot performs the following tasks:

1. Reads the DTB from the `kernel-dtb` partition, if it exists.

2. Modifies the DTB to add device tree nodes that are specific to the platform.
3. Passes the modified DTB to the kernel, if Quickboot initially read the DTB from the kernel-dtb partition.

12.1.6.1 To flash a custom DTB file

- > Flash the file with the bootburn -u option and provide the path to your customized DTB file.

The default path is `tegra186-<board>-010-a01-00-base.dtb`.

12.1.7 Configuring Device Tree Support

By default, the board configuration files enable device tree support for a DTB partition device. The `TARGET_KERNEL_DT_NAME` configuration contains the prefix for the DTB/DTSI files for your platform.

12.1.8 Device Tree Data Format

The device tree data format represents information for the devices on a board. It has these characteristics:

- > A tree structure of nodes and properties
- > A single root node "/"
- > Child nodes with properties
- > Key-value pairs represent properties
- > Keys used as property identifiers
- > Values are empty or byte streams
- > Data types of streams:
 - Strings "Value"
 - List of strings: "Value_1", "Value_2"
 - Binary data: [0x1 0x2 0x3]
 - Cell: <0x1 0x2 0x3>

The device tree data format is part of the Open Firmware industry standard (IEEE 1275).

12.1.9 SW-EDID

SW-EDID is a mechanism to provide EDID as a blob in a DTB. The driver reads the EDID data from the device tree instead of from the panel or monitor. SW-EDID is useful in the following cases:

- > The DDC channel of HDMI is not connected and another method is required to provide mode information to the driver. Instead of hard-coding the board file, the EDID blob is supplied in the DTB for the driver to read at runtime.

- > Debugging or experimental purposes where there is a need to override the EDID.
- SW-EDID supports HDMI, DP, and LVDS drivers.

12.1.10 Device Tree Binding Document

The device tree binding (DTB) document provides information about device tree properties supported by the kernel device tree binding documents, which are available as part of the Linux kernel source.

Device tree binding documentation is in the Linux kernel source released with the package at the following path:

```
kernel/Documentation/devicetree/bindings/
nvidia/Documentation/devicetree/bindings/
```

12.2 Device Tree Cleaner (delnode)

The `delnode.sh` tool removes disabled device tree nodes from Device Tree Blob (DTB) files that are used by the system. Removing disabled device tree node reduces the file sizes of the DTB files, which in turn speeds up the boot process. The speed up happens because optimized DTB files are faster to decompress and load.

12.2.1 Calling delnode

- > From the Foundation or Linux build paths, enter:

```
delnode.sh <dtb_file> <build_dir> <dtc_compiler_to_use>
```

Where the arguments specify:



Note:

`delnode.sh` is invoked as part of bind makefile.

12.2.2 Example

The following example shows DTBs (converted to Device Tree Source (DTS) version) before and after `delnode` processes them.

12.2.2.1 Before

```
usb3 {
    lanes {
        usb3-0 {
            status = "okay";
            #phy-cells = <0x0>;
```

```

        nvidia,function = "xusb";
        linux,phandle = <0x94>;
        phandle = <0x94>;
    };
    usb3-1 {
        status = "disabled";
        #phy-cells = <0x0>;
    };
    usb3-2 {
        status = "disabled";
        #phy-cells = <0x0>;
        nvidia,function = "xusb";
    };
    usb3-3 {
        status = "disabled";
        #phy-cells = <0x0>;
    };
}

```

12.2.2.2 After

```

usb3 {
    lanes {
        usb3-0 {
            status = "okay";
            #phy-cells = <0x0>;
            nvidia,function = "xusb";
            linux,phandle = <0x94>;
            phandle = <0x94>;
        };
    };
}

```

12.3 Terminal Emulation

With utilities PuTTY or Minicom, you can communicate from the host machine to the *target* board over a serial line. NVIDIA recommends using the Minicom terminal emulation program for better display of the setup prompts, but information is also provided on options to improve display of setup prompts.

It is recommended to use the [tcu_muxer utility](#) to demultiplex the console output from the virtualized processors to be able to connect to the Guest OS and other VM consoles. If the tcu_muxer is not used, it is possible that input at the console may be corrupted.

12.3.1 About Minicom

[Minicom](#) is a serial communication program that enables an admin user to communicate using the serial port. In DRIVE OS SDK, we use Minicom to communicate from host machine to DRIVE platform.

12.3.1.1 Configuring Minicom

This topic explains how to modify the Minicom configuration file for the USB port and serial ID for your device. The default serial port settings depend on your development board. If your port differs from the default, you must modify the Minicom configuration file for your device (procedure below).



Note:

Although the Minicom configuration file contains the following statement, it is recommended that you manually edit the file for your changes. This approach is less error-prone than using the Minicom -s option.

```
# Machine-generated file - use "minicom -s" to change parameters.
```

Port Settings

By default, the serial port settings are shown below. You must modify the port to the actual device for the serial port.

```
port      /dev/ttys0
baudrate 115200
bits      8
parity    N
stopbits  1
rtscts    No
xonxoff   No
```

12.3.1.2 Running Minicom

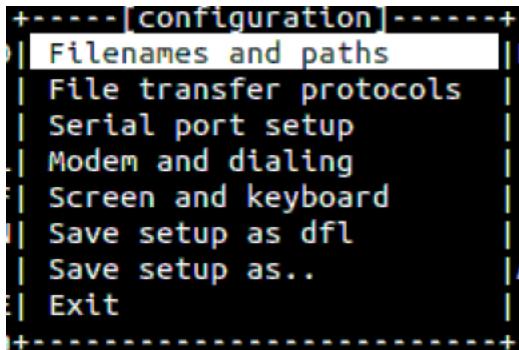
12.3.1.2.1 Prerequisites

- > You have modified the Minicom configuration file for your device's USB port and serial ID.
- > You have connected the device to your host system.
- > The device is on and awake.
- > The hardware flow control is turned off.

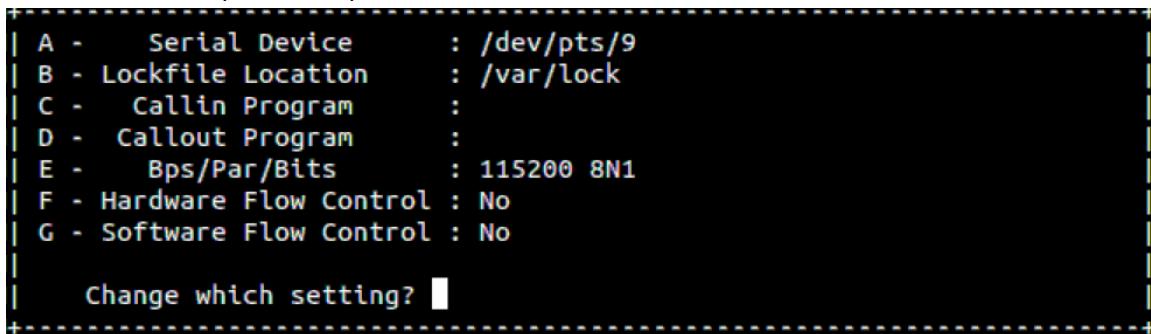
12.3.1.2.2 To configure Minicom

1. In the Minicom console, enter the following command (without spaces between the letters):

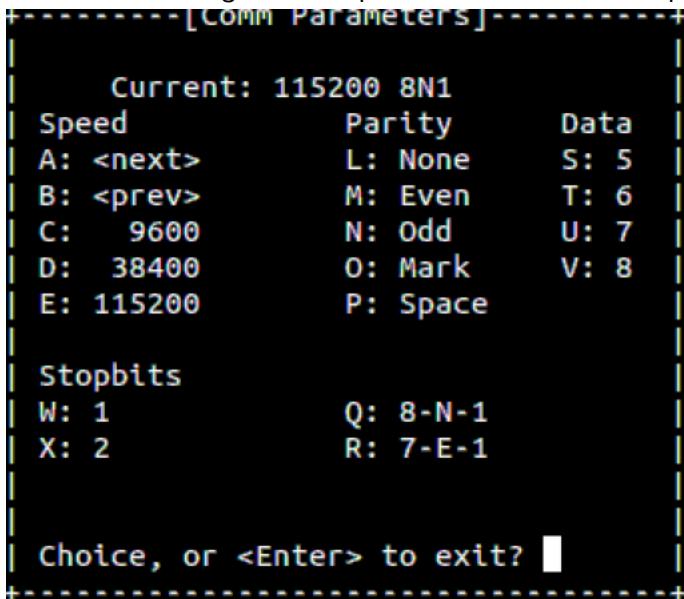
```
CTRL-A 0
```



2. Scroll to "Serial port setup" and hit ENTER.



3. Enter A to enter the serial Device number.
4. Enter E to configure the speed and other comm parameters of the port.



5. When finished, hit ENTER to exit the screen. Hit ENTER again if there are no further changes to the serial port setup screen, and then scroll down to Exit to exit the configuration screen.

12.3.1.2.3 Determining the USB Port and Serial ID

This topic explains how to get the USB port number and serial number port settings of your *target* device. Use this information to configure Minicom, depending on your product.

On Linux hosts, USB serial ports typically appear as:

```
/dev/ttyACM<number> or /dev/ttyUSB<number>
```

Where <number> is the port number.

Applies to: Releases supporting Windows:

A host machine may contain internal hardware for serial ports that have no external connector. So, if the host machine contains one normal serial connector in the back of the machine and one hidden internal serial port, then the serial port on the back can appear as /dev/ttYS2 not /dev/ttYS1.

The device USB port numbers and serial IDs are as follows:

	Serial ID	USB Port
AURIX safety MCU	P3710	/dev/ttyACM1
	P3663	/dev/ttyUSB1
Orin debug UART	P3710	/dev/ttyACM0
	P3663	/dev/ttyUSB0

For Orin debug UART, use the output of the [tcu muxer utility](#) to get the virtual console for the Guest OS. As shown in the following example, the Guest OS console is /dev/pts/40.

```
/dev/pts/7 RCE
/dev/pts/9 FSI
/dev/pts/10 PSCFW
/dev/pts/11 DCE
/dev/pts/30 BPMP
/dev/pts/33 SCE
/dev/pts/34 SPE
/dev/pts/38 TZ
/dev/pts/40 CCPLEX: 0
/dev/pts/41 CCPLEX: 1
/dev/pts/43 CCPLEX: 2
/dev/pts/44 CCPLEX: 3
/dev/pts/45 CCPLEX: 4
/dev/pts/46 CCPLEX: 5
/dev/pts/48 CCPLEX: 6
/dev/pts/49 CCPLEX: 7
/dev/pts/50 CCPLEX: 8
/dev/pts/59 CCPLEX: 9
/dev/pts/60 CCPLEX: 10
```

```
/dev/pts/61 CCPLEX: 11
#####
## 001 ##
/dev/pts/40 [Guest VM 0]
/dev/pts/41 [Update service]
/dev/pts/43 [Resource Manager Server]
/dev/pts/44 [Storage Server]
/dev/pts/45 [sysmgr]
/dev/pts/46 [bpmp_server_native]
/dev/pts/48 [se_server_native]
/dev/pts/61 [Hypervisor]
```

For all ports, the settings should be as follows:

```
baudrate 115200
bits     8
parity   N
stopbits 1
rtscts   No
xonxoff  No
```

12.3.1.2.4 To run Minicom

- > In a shell window, enter the following command:

```
sudo minicom -w -D /dev/pts/<number> -R utf8 -t xterm
```

The `-w` option enables line-wrapping, `-R utf8` enables UTF-8 character set.

12.3.1.2.5 Toggling the Line Wrap Setting on Minicom

The Minicom command line in [Running Minicom](#) includes the `-w` option, which enables line wrapping. You can use the `W` configuration setting to change that behavior.

To toggle line wrap on/off

- > In the Minicom console, enter the following command (without spaces between the letters):

```
CTRL-A Z W
```

12.3.1.2.6 Toggling the Line Feed Setting on Minicom

By default, Minicom enables line-wrapping. With that option, Minicom adds a line feed before every carriage return displayed on the screen.

Some terminals, have poor attribute handling (serial instead of parallel). If you are using such a terminal, you must start Minicom with the following option:

```
-a off
```

Or you must toggle off the line feed, as described in the following instructions.

To toggle line feed on/off

- In the Minicom console, enter the following command:

```
CTRL-AZA
```

12.3.2 About PuTTY

[PuTTY](#) is both a serial communication program and a terminal emulator that uses the SSH or TELNET protocol. PuTTY supports the full UTF-8 character-set.

12.3.2.1 Installing PuTTY

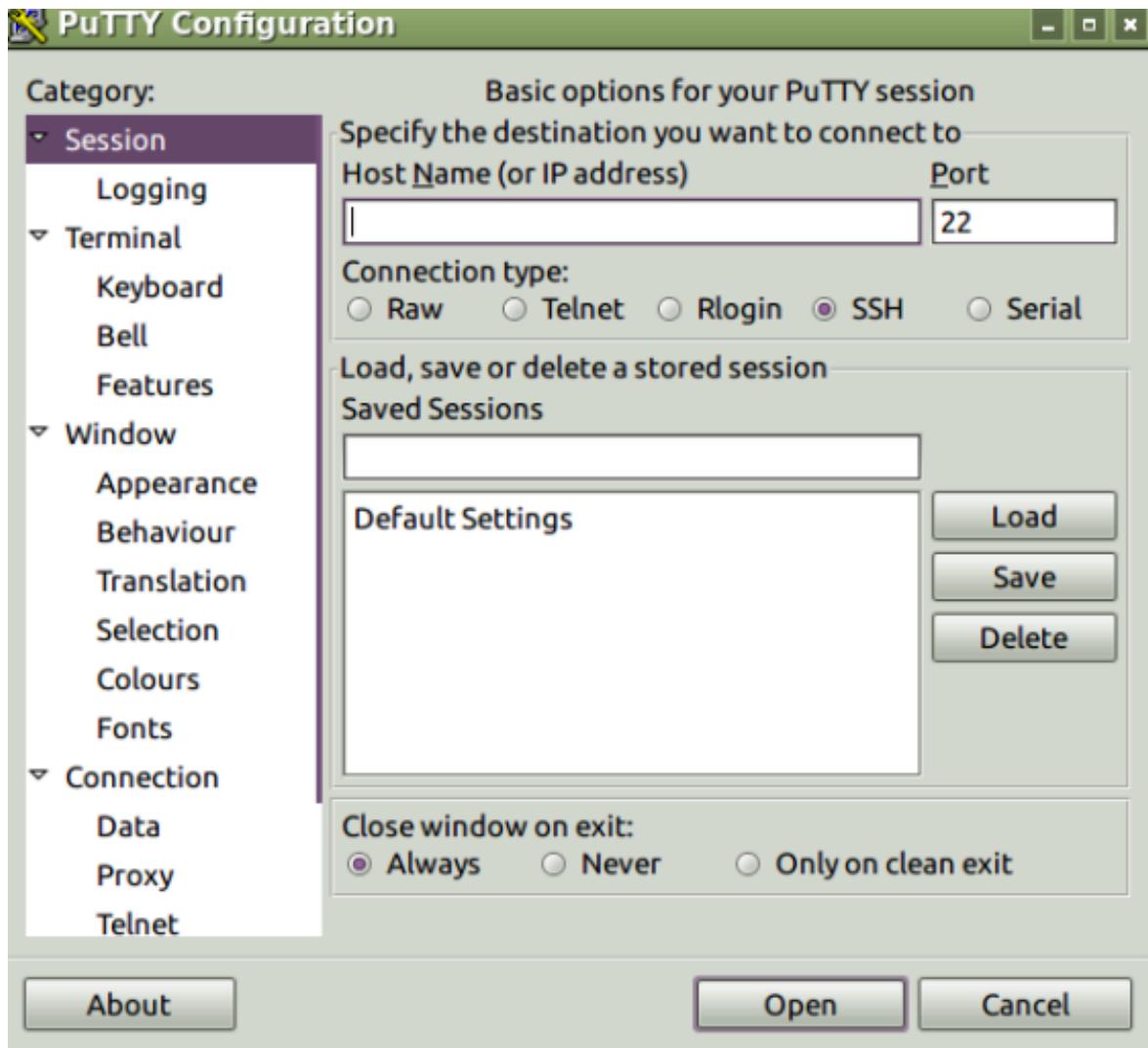
PuTTY is installable via apt-get from Canonical mirrors. Use the following command to install PuTTY:

```
$ sudo apt-get install putty
```

12.3.2.2 PuTTY: Connect to Serial Port

Before connecting to a serial port, you must start PuTTY as a superuser (or as root user) using following command. This command opens the putty configuration dialog box.

```
$ sudo putty
```



In the above dialog box:

1. Choose Connection type as Serial.
2. In Serial line enter the USB Serial port path obtained above /dev/ttyACM<number> or /dev/ttyUSB<number> or /dev/pts/<number>.
3. Enter Speed to 115200.
4. Click Open

Steps 1-3 opens a new window to receive serial data from the DRIVE platform.

12.3.2.3 PuTTY: Save and Load Sessions

PuTTY allows users to save entered configuration into session entry which can be loaded later.

To save a session, enter the terminal settings, as noted in the previous topic, from steps 1-3, and enter a session name in the Saved Sessions field.

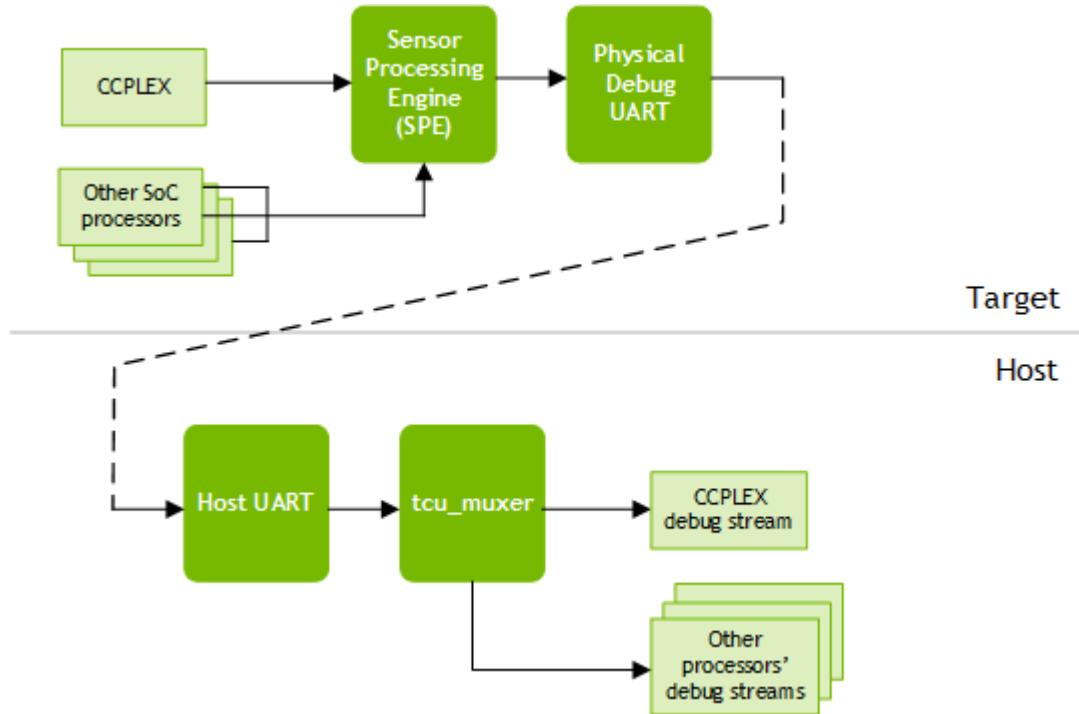
When PuTTy launches, it lists saved sessions. Slick the choice of session, click Load, and then click Open.

12.4 Tegra Combined UART and the `tcu_muxer` Utility

The Tegra Combined UART (TCU) is a system that multiplexes debug information from the processors in the CCPLEX cluster with information from other processors. The multiplexing is accomplished in the Sensor Processing Engine (SPE), but involves all of the processors that supply information.

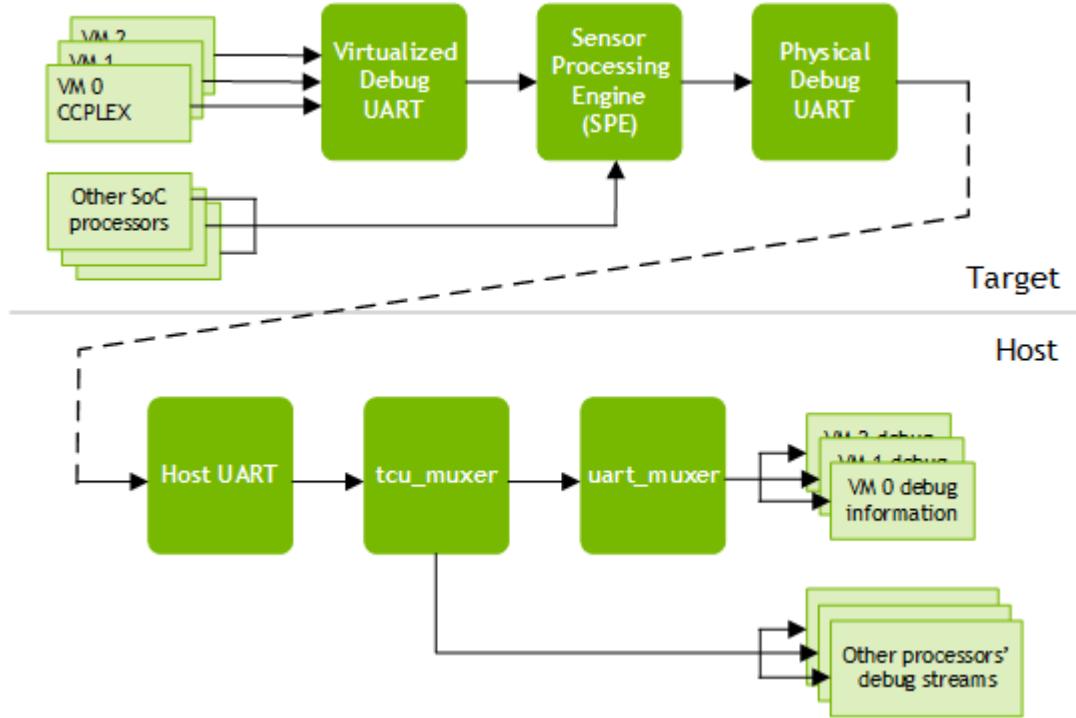
`tcu_muxer` is a utility which runs on a host system and demultiplexes the debug information multiplexed by the TCU.

This diagram shows the relationship of the components for a native (non-virtualized) target.



On a virtualized target, the Virtualized Debug UART multiplexes debug information from the VMs in the CCPLEX cluster and passes the multiplexed data stream to the TCU. The TCU multiplexes it with debug information from the other R5 components. On the host, the CCPLEX debug information customarily is passed from `tcu_muxer` to `uart_muxer`, which demultiplexes the individual VMs' debug information.

This diagram shows the relationship of the components in a virtualized system.



12.4.1 tcu_muxer Tool

12.4.1.1 Usage

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer -g <guest_nr> -b <hyp_nr> -d <device>
```

This table describes the command line options recognized by **tcu_muxer**:

Command line option	Meaning
-h	Prints this help screen.
-i	Enables the patch for line ending.
-u	Use separate uart_muxer tool for Guest console.
-g <int>	Spawns <int> consoles for Hypervisor. Defaults to 1.
-d <dev>	Specifies the device name of the host UART that receives debug data from the target. Defaults to /dev/ttyUSB3.
-r <rate>	Specifies data rate of the UART in bits/second. Defaults to 115200.

Command line option	Meaning
-b <int>	Print Hypervisor broadcast only on console <int>. This value starts from 0 (zero).
-s <path>	Saves the output to a directory <path>.
-l <path>	Saves the raw output with tags to a log file <path>

12.4.1.2 Output

tcu_muxer displays pseudo terminal number mapping to different CPU clusters. All messages from these CPU clusters are redirected to the corresponding pseudo terminal.

tcu_muxer and the helper scripts are located at:

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/
```

12.4.2 tcu_muxer Usage in NVIDIA Native OS System

This creates multiple pseudo terminals for each R5 and CCPLEX. Access Native OS shell through the terminal corresponding to CCPLEX.

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer -g <guest_nr> -b <hyp_nr> -d
<device>
/dev/pts/52 RCE
/dev/pts/55 BPMP
/dev/pts/56 SCE
/dev/pts/57 SPE
/dev/pts/58 TZ
/dev/pts/59 CCPLEX: 0
```

12.4.3 tcu_muxer Usage in NVIDIA Virtualization System

Virtualization System executing on CCPLEX cluster additionally has multiple UART streams corresponding to partition(s) running over Hypervisor. The tcu_muxer tool supports Virtualization and allows creation of pseudo terminals for each of the partition.

12.4.3.1 Usage

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer -g <guest_nr> -b <hyp_nr> -d
<device>
```

Where:

- > guest_nr: represents the number of virtual machines

- > hyp_nr: represents the pseudo terminal for Hypervisor

Depending on the requirement, the tcu_muxer tool can be launched in different modes:

12.4.3.2 Usage mode #1

In this mode, tools similar to minicom are used to attach to each of the UART terminal.

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer -g 11 -b 10 -d /dev/ttyACM0
Opening: /dev/pts/16 RCE
Opening: /dev/pts/17 BPMP
Opening: /dev/pts/18 SCE
Opening: /dev/pts/19 SPE
Opening: /dev/pts/20 TZ
Opening: /dev/pts/22 CCPLEX: 0
Opening: /dev/pts/23 CCPLEX: 1
Opening: /dev/pts/24 CCPLEX: 2
Opening: /dev/pts/25 CCPLEX: 3
Opening: /dev/pts/26 CCPLEX: 4
Opening: /dev/pts/27 CCPLEX: 5
Opening: /dev/pts/28 CCPLEX: 6
Opening: /dev/pts/29 CCPLEX: 7
Opening: /dev/pts/30 CCPLEX: 8
Opening: /dev/pts/31 CCPLEX: 9
Opening: /dev/pts/32 CCPLEX: 10
#####
/dev/pts/22 [Guest 0]
/dev/pts/23 [BPMP]
/dev/pts/24 [Resource Manager]
/dev/pts/25 [Monitor Partition]
/dev/pts/26 [System Manager]
/dev/pts/27 [Storage]
/dev/pts/28 [Security Engine]
/dev/pts/29 [Debug Server]
/dev/pts/30 [TrustZone Server]
/dev/pts/32 [Hypervisor]
```

CCPLEX: 0 corresponds to VM0, CCPLEX: 1 corresponds to VM1 and so on. CCPLEX:10 corresponds to Hypervisor as mentioned by -b switch.

12.4.3.3 Usage mode #2 (terminal logging)

In this mode, output from all the terminals is logged in their respective log file.

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer -g 11 -b 10 -d /dev/ttyACM0 -s /
path/to/logging/dir/
Opening: /dev/pts/16 RCE
Opening: /dev/pts/17 BPMP
Opening: /dev/pts/18 SCE
Opening: /dev/pts/19 SPE
Opening: /dev/pts/20 TZ
Opening: /dev/pts/22 CCPLEX: 0
Opening: /dev/pts/23 CCPLEX: 1
```

```

Opening: /dev/pts/24 CCPLEX: 2
Opening: /dev/pts/25 CCPLEX: 3
Opening: /dev/pts/26 CCPLEX: 4
Opening: /dev/pts/27 CCPLEX: 5
Opening: /dev/pts/28 CCPLEX: 6
Opening: /dev/pts/29 CCPLEX: 7
Opening: /dev/pts/30 CCPLEX: 8
Opening: /dev/pts/31 CCPLEX: 9
Opening: /dev/pts/32 CCPLEX: 10
#####
##### 001 ##
/dev/pts/22 [Guest 0]
/dev/pts/23 [BPMP]
/dev/pts/24 [Resource Manager]
/dev/pts/25 [Monitor Partition]
/dev/pts/26 [System Manager]
/dev/pts/27 [Storage]
/dev/pts/28 [Security Engine]
/dev/pts/29 [Debug Server]
/dev/pts/30 [TrustZone Server]
/dev/pts/32 [Hypervisor]
$ ls /path/to/logging/dir/
BPMP.txt CCPLEX10.txt CCPLEX2.txt CCPLEX4.txt CCPLEX6.txt CCPLEX8.txt raw-logs.txt
SCE.txt TZ.txt
CCPLEX0.txt CCPLEX1.txt CCPLEX3.txt CCPLEX5.txt CCPLEX7.txt CCPLEX9.txt RCE.txt SPE.txt

```

Each file contains the respective terminal's output.

12.4.3.4 Usage mode #3 (raw logging)

In this mode, a raw file is additionally created that contains the output of all the terminals.

```

$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer -g 11 -b 10 -d /dev/ttyACM0 -l /
path/to/logging/file
Opening: /dev/pts/16 RCE
Opening: /dev/pts/17 BPMP
Opening: /dev/pts/18 SCE
Opening: /dev/pts/19 SPE
Opening: /dev/pts/20 TZ
Opening: /dev/pts/22 CCPLEX: 0
Opening: /dev/pts/23 CCPLEX: 1
Opening: /dev/pts/24 CCPLEX: 2
Opening: /dev/pts/25 CCPLEX: 3
Opening: /dev/pts/26 CCPLEX: 4
Opening: /dev/pts/27 CCPLEX: 5
Opening: /dev/pts/28 CCPLEX: 6
Opening: /dev/pts/29 CCPLEX: 7
Opening: /dev/pts/30 CCPLEX: 8
Opening: /dev/pts/31 CCPLEX: 9
Opening: /dev/pts/32 CCPLEX: 10
#####
##### 001 ##
/dev/pts/22 [Guest 0]
/dev/pts/23 [BPMP]

```

```
/dev/pts/24 [Resource Manager]
/dev/pts/25 [Monitor Partition]
/dev/pts/26 [System Manager]
/dev/pts/27 [Storage]
/dev/pts/28 [Security Engine]
/dev/pts/29 [Debug Server]
/dev/pts/30 [TrustZone Server]
/dev/pts/32 [Hypervisor]
```

/path/to/logging/file contains the raw file with the control characters that can be demuxed offline using the \$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer_raw_log_dump.py script.

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer_raw_log_dump.py -h
Usage: ./tcu_muxer_raw_log_dump.py -l path [-g|-c]
      -l, --logfile : Path of the file containing the raw logs captured using tcu_muxer -l option
      -g, --guestid : Guest ID. Prints Hypervisor log if no value is passed
      -c, --cluster : Cluster Name. One of: "RCE", "BPMP", "SCE", "SPE", "TZ", "CCPLEX"
```

The dumps the BPMP logs on the stdout extracted from /path/to/logging/file:

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer_raw_log_dump.py -l /path/to/logging/file -c BPMP
```

Dump the CCPLEX: 5 logs on the stdout extracted from /path/to/logging/file:

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer_raw_log_dump.py -l /path/to/logging/file -g 5
```

Use the raw log file to see the timeline of events across terminals and to represent the state of the system in a single file to allow the recipient to extract the logs they are interested in.

The -s and -l option can be used together.

12.4.3.5 Usage mode#4 (tmux with tcu_muxer)

This is a wrapper over the tcu_muxer tool that creates a tmux session with a separate window for each terminal.

```
$<top>/drive-foundation/tools/muxer/tcu_muxer/tcu_muxer_tmux.sh -g 11 -b 10 -d /dev/ttyACM0 -s /path/to/logging/dir/ -l /path/to/logging/file
Opening: /dev/pts/16 RCE
Opening: /dev/pts/17 BPMP
Opening: /dev/pts/18 SCE
Opening: /dev/pts/19 SPE
Opening: /dev/pts/20 TZ
Opening: /dev/pts/22 CCPLEX: 0
Opening: /dev/pts/23 CCPLEX: 1
Opening: /dev/pts/24 CCPLEX: 2
Opening: /dev/pts/25 CCPLEX: 3
Opening: /dev/pts/26 CCPLEX: 4
Opening: /dev/pts/27 CCPLEX: 5
Opening: /dev/pts/28 CCPLEX: 6
```

```

Opening: /dev/pts/29 CCPLEX: 7
Opening: /dev/pts/30 CCPLEX: 8
Opening: /dev/pts/31 CCPLEX: 9
Opening: /dev/pts/32 CCPLEX: 10
##### 001 ##
/dev/pts/22 [Guest 0]
/dev/pts/23 [BPMP]
/dev/pts/24 [Resource Manager]
/dev/pts/25 [Monitor Partition]
/dev/pts/26 [System Manager]
/dev/pts/27 [Storage]
/dev/pts/28 [Security Engine]
/dev/pts/29 [Debug Server]
/dev/pts/30 [TrustZone Server]
/dev/pts/32 [Hypervisor]
$ tmux attach

```

Attaches to the tmux session. There are two windows: one for all the R5 clusters and the other for all the CCPLEX terminals. Select the panes and windows using the mouse. To select text, hold shift and select; otherwise, the selection is captured by tmux and is not copied to the system clipboard.

12.4.4 Finding the Number of VM Partitions

`tcu_muxer` takes the number of terminals to launch as an input parameter, `-g`. The required number of terminals depends on the number of virtual machines/partitions present in a given virtualized system configuration.

The tool `pctdump` can get this information by reading the configuration blob.

12.4.4.1 Usage

```
pctdump <pct configuration blob>
```

Here is an example (for P3710-01/P3663 T19x Hypervisor Linux Configuration):

```

$ <TOP>/tools/pctdump/pctdump <TOP>/virtualization/
hypervisor/t19x/configs/release/pct/linux-linux/pct.bin
Guest Names
[0] Guest 0
[1] BPMP
[2] Resource Manager
[3] Monitor Partition
[4] System Manager
[5] Storage
[6] Security Engine

```

To the total of 7 VMs, add 2 VMs for internal debug and TZ server and 1 VM for Hypervisor. The final count is 10, and the `tcu_muxer` command is:

```
$ tcu_muxer -g 10 -b 9 -d /dev/ttyACM0
```

12.5 tegrastats Utility

This SDK provides the `tegrastats` utility, which reports memory usage and processor usage for Tegra-based devices.

You can find the utility in your package at the following location.

```
<top>/drive-linux/filesystem/contents/bin/tegrastats
```

12.5.1 Reported Statistics

There are two types of thermal sensors:

- SOC_THERM sensors
- tmp451 sensors

CPU / GPU / CV / SoC thermal sensors are the on-chip SOC_THERM thermal sensors, which are controlled by BPMP firmware and `tegra-bpmp-thermal` Linux kernel driver. EXT0, EXT1 thermal sensors are tmp451 sensors, which are from TEXAS INSTRUMENTS. In K5.10, they are controlled by `nct1008` Linux kernel driver. In K5.15, the driver is switched to `lm90` Linux kernel driver. There are both local and remote EXT0, EXT1 sensors, where a local sensor means that the sensor is located on the board and a remote sensor means that the sensor is located on the SoC.

The following table shows the statistics that the `tegrastats` utility reports.

Statistic	X	Y	Z
<code>RAM X/Y (lfb NxZ)</code> Largest Free Block (lfb) is a statistic about the memory allocator. It refers to the largest contiguous block of physical memory that can be allocated: at most, 4 MB. It can become smaller with memory fragmentation. The physical allocations in virtual memory can be bigger.	Amount of RAM in use in MB.	Total amount of RAM available for applications.	Z is the size of the largest free block, N the number of free blocks of this size.
<code>CPU [X%, Y%, ,]@Z</code> or <code>CPU [X%@Z, Y%@Z, ...]</code> X and Y are rough approximations based on time spent in the system idle process as reported by the Linux kernel in <code>/proc/stat</code> .	Load statistics for each of the CPU cores relative to the current running frequency Z, or 'off' in case a core is currently powered down.	Load statistics for each of the CPU cores relative to the current running frequency Z, or 'off' in case a core is currently powered down.	CPU frequency in megahertz. Goes up or down dynamically depending on the CPU workload.
<code>GR3D_FREQ X%@[Y, Y, ...]</code> GR3D is the iGPU engine.	Percent of the GR3D that is being used, relative to the current running frequency. Aggregated between GPC0 and GPC1.	GR3D frequency in megahertz for each available GPC (that is, GPC0, GPC1, and so on.)	N/A
<code>X1@Y1C X2@Y2C X3@Y3C...</code> X1, X2, X3 denote the list of the available sensors	The sensor name	The sensor's temperature in Celsius	N/A
<code>EMC_FREQ @Y</code> NVIDIA DRIVE OS Linux SDK Developer Guide EMC is the external memory controller.	N/A	EMC frequency in megahertz	N/A

12.5.2 Running tegrastats

When you run `tegrastats` on Linux devices, it prints statistics to `stdout`.

12.5.2.1 Example Log Print

```
RAM 640/31318MB (1fb 7604x4MB) SWAP 0/15659MB (cached 0MB) CPU
[0%@2296, 0%@2295, 0%@2297, 0%@2300, 0%@2232, 0%@2287, 0%@2297, 0%@2298, 0%@2287, 0%@2298, 0%@2299, 0%@2291]
EMC_FREQ 0%@1600 GR3D_FREQ 0%@0 GR3D2_FREQ 0%@0 NVJPG1 1100 VIC_FREQ 1164 APE 233
CV0@-256C CPU@41.531C Tdiode@29.25C SOC2@37.093C SOC0@38.625C CV1@-256C GPU@-256C
tj@41.531C SOC1@38.406C CV2@-256C VDD_GPU 0mW/0mW VDD_CPU 1589mW/1589mW VDD_SOC
5134mW/5134mW VDD_CV 0mW/0mW VDDQ_VDD2_1V8AO 682mW/682mW VIN_SYS_5V0 4421mW/4421mW
```

12.5.2.2 To run tegrastats



Note: Run the utility as root to display all enabled stats.

- > To run `tegrastats` in the background, execute the following command:

```
tegrastats --interval <int> --logfile <out_file> &
```

Where:

- `<int>` is the interval between log prints in milliseconds.
- `<out_file>` is the pathname of the output file to which `tegrastats` writes the log prints.

- > To run `tegrastats` in the foreground, omit the trailing '&'. You may also omit the `--logfile` option to allow log output to go to `stdout`:

```
tegrastats --interval <int>
```

12.5.2.3 To stop tegrastats

- > If `tegrastats` is running in the background, execute the following commands:

```
ps
kill -9 <pid>
```

Where `<pid>` is the process ID of `tegrastats` as reported by the `ps` command.

Alternatively, you may run:

```
tegrastats --stop
```

- > If `tegrastats` is running in the foreground, press `CTRL+C` in the window where it is running.

12.5.3 Re-Deploying tegrastats

The `tegrastats` utility is preinstalled. If you have removed `tegrastats` from the build, you can re-deploy it on the target at runtime.

12.5.3.1 To re-deploy tegrastats

- Execute the following command from the host PC:

```
scp tegrastats nvidia@<TARGET_DEVICE_IP>:/home/nvidia/
```

12.5.4 tegrastats Options

tegrastats supports the following command line options.

Option	Meaning
--help	Prints this help screen.
--interval <millisec>	Samples the information in <milliseconds>.
--logfile <filename>	Dumps the output of tegrastats to <filename>.
--load_cfg <filename>	Loads the information from <filename>.
--readall	Collects all stats, including performance intensive stats.
--save_cfg <filename>	Saves the information to <filename>.
--start	Runs tegrastats as a daemon process in the background.
--stop	Stops any running instances of tegrastats.
--verbose	Prints verbose message.

12.6 Benchmarking Library

This page captures the documentation about the NvPlayfair benchmarking helper library.

NvPlayfair benchmarking library provides helper methods to NVIDIA DRIVE® OS benchmarks to facilitate various benchmarking related operations such as timestamping, data recording, rate-limiting, and report generation.

In NVIDIA DRIVE OS SDKs, the following components are provided for the benchmarking library:

- **nvplayfair.h:** The header file contains definitions of the public interfaces and data-structures of the benchmarking library. It can facilitate recompilation of benchmarks in the SDK, which make use of the library. The header file is present in the "include" folder in the SDK installation directory.

- **libnvpplayfair.so:** The library itself is distributed as a compiled shared object file. After the relevant Debian package is installed and the file system is rebuilt, the library can be found in the `lib-target` folder in the SDK installation directory and it is automatically copied to the `/usr/lib` location in the target file system during flashing.

12.6.1 API Documentation

This section contains documentation about the data-structures, macros, and public interfaces of NvPlayfair benchmarking library.

Data Structures

NvpStatus_t

This enum defines the possible return status of the NvPlayfair library APIs.

```
typedef enum {
    NVP_PASS,
    NVP_FAIL_ALLOC,
    NVP_FAIL_NOINIT,
    NVP_FAIL_FILEOP,
    NVP_FAIL_NULLPTR,
    NVP_FAIL_NO_SAMPLES,
    NVP_FAIL_VERSION_MISMATCH,
    NVP_FAIL_INVALID_TIME_UNIT,
    NVP_FAIL_INVALID_LOG_BACKEND,
    NVP_FAIL_SAMPLE_COUNT_MISMATCH
} NvpStatus_t;
```

NvpTimeUnits_t

This enum defines the time-units understood by relevant NvPlayfair library APIs.

```
typedef enum {
    SEC,
    MSEC,
    USEC,
    NSEC
} NvpTimeUnits_t;
```

NvpLogBackend_t

This enum defines the backend, which can be the receiver of output from relevant library APIs.

```
typedef enum {
    CONSOLE,
    NVOS
} NvpLogBackend_t;
```

NvpPerfStats_t

This data-structure is used to record various statistics about the respective latency data.

```
typedef struct {
    double    min;
    double    max;
    double    mean;
    double    pct99;
    double    stdev;
    uint32_t  count;
} NvpPerfStats_t;
```

NvpRateLimitInfo_t

This data-structure defines a rate-limit object that can be used to make a benchmark repeat its compute loop at a given frequency.

```
typedef struct {
    uint32_t periodUs;
    uint32_t periodNumber;
    uint64_t periodicExecStartTimeUs;
} NvpRateLimitInfo_t;
```

NvpPerfData_t

This is the main data-structure provided by the NvPlayfair library. It can be configured as a benchmarking data object that can be used to store all the latency data gathered for a particular metric by a benchmark.

```
typedef struct {
    uint64_t sampleNumber;
    uint64_t *timestamps;
    uint64_t *latencies;
    uint32_t maxSamples;
    bool     initialized;
    char     *filename;
} NvpPerfData_t;
```

NvpLibVersion_t

This data structure is used to capture the version information of the library.

```
typedef struct {
    uint64_t major;
    uint64_t minor;
} NvpLibVersion_t;
```

12.6.2 Macros

This section describes benchmarking library macros.

DISABLE_NVPLAYFAIR

A benchmark can define this macro as part of the CFLAGS given to the compiler. This will transform all the calls to NvPlayfair library APIs into innocuous macros and remove the dependency of the benchmark on libnvplayfair.so during runtime.

NVP_CHECKERR_EXIT

This is an error checking macro, which can be used as a wrapper around all NvPlayfair library API calls. It provides straight-forward error handling by printing a message on console / log file and exiting the program when an error is reported by an NvPlayfair API.

```
/* Example Usage */
NVP_CHECKERR_EXIT(NvpRecordSample(NULL, startTimeMark, endTimeMark));

/* Expected Output
 * myBenchmark.c, myFunction:13, NvPlayfair Error: NVP_FAIL_NULLPTR */
```

NVP_GET_SAMPLE_COUNT

This macro can be used to get the sample count in the internal ring-buffer of the given NvpPerfData_t* object.

```
/* Example Usage */
uint64_t curSampleCount = NVP_GET_SAMPLE_COUNT(perfDataObj);
```

for_each_sample

This is a convenience macro, which can be used to iterate over the timestamp and latency values of all the samples recorded so far in the given NvpPerfData_t* object.

```
/* Example Usage */
uint32_t sampleNumber = 0;
uint64_t timestamp, latency;

for_each_sample(&perfData, timestamp, latency) {
    printf("Sample #: %d Timestamp: %ul Latency: %ul\n",
           sampleNumber++, timestamp, latency);
}
```

for_each_sample_latency

This is similar to "for_each_sample" macro but it provides only the latency value for each sample in the given object.

```
/* Example Usage */
uint32_t sampleNumber = 0;
uint64_t latency;

for_each_sample_latency(&perfData, latency) {
    printf("Sample #: %d Latency: %ul\n", sampleNumber++, latency);
}
```

12.6.3 Functions

This section describes benchmarking library functions.

NvpGetTimeMark

This API returns an opaque timestamp in a safe and efficient manner under out-of-order execution in the target platform. The given timestamp is to be understood by NvPlayfair library only.

```
/* Inputs
 *   None
 *
 * Output
 *   uint64_t  Opaque timestamp value
 */
static inline uint64_t
NvpGetTimeMark(void);

/* Example Usage */
uint64_t timeMark;

timeMark = NvpGetTimeMark();
```

NvpConvertTimeMarkToNsec

This API converts the opaque time-mark value to nsec timestamp. Note that, in ARM architecture, the function assumes that 1-cycle = 32-nsec which is true for the Tegra counters running at the frequency of 31.25-MHz.

```
/* Inputs
 *   timeMark  64-bit variable containing the opaque time-mark value
 *
 * Output
 *   uint64_t  Flat timestamp value in nsec
 */
static inline uint64_t
NvpConvertTimeMarkToNsec(uint64_t timeMark);

/* Example Usage */
uint64_t timestamp_ns;

timestamp_ns = NvpConvertTimeMarkToNsec(NvpGetTimeMark());
```

NvpConstructPerfData

This is the primary function for populating a given `NvpPerfData_t*` object. It allocates necessary memory for internal buffers based on the given number of samples and stores the given information for internal book-keeping.

```
/* Inputs
 *   perfData          An NvpPerfData_t* object
 *   numSamples         An integer specifying the total number of samples
 *                     that can be stored in the NvpPerfData_t* object
```

```

/*      filename           A character string specifying the name of a file
*                           which can be used to save the performance data
*
* Output (NvpStatus_t)
*   NVP_PASS             The API call completed successfully
*   NVP_FAIL_NO_SAMPLES Call failed because the numOfSamples was zero
*   NVP_FAIL_NULLPTR    Call failed because one of the input pointers was NULL
*   NVP_FAIL_ALLOC      Call failed because the library could not allocate memory
*/
NvpStatus_t
NvpConstructPerfData(NvpPerfData_t *perfData,
                     uint32_t      numOfSamples,
                     char const    *filename);

/* Example Usage */
NvpPerfData_t perfData;
uint32_t numOfSamples = 1000U;
char const *filename = "myLatencyData.csv";

NVP_CHECKERR_EXIT(NvpConstructPerfData(&perfData, numOfSamples, filename));

```

NvpDestroyPerfData

This is a complementary function to NvpConstructPerfData; it can be used to free up all resources associated with the given NvpPerfData_t* object. It should be called once the user is done with the object.

```

/* Inputs
*   perfData            An NvpPerfData_t* object
*
* Output (NvpStatus_t)
*   NVP_PASS             The API call completed successfully
*   NVP_FAIL_NULLPTR    Call failed because the input pointer was NULL
*/
NvpStatus_t
NvpDestroyPerfData(NvpPerfData_t *perfData);

/* Example Usage */
NVP_CHECKERR_EXIT(NvpDestroyPerfData(&perfData));

```

NvpRecordSample

This is a complementary function to NvpConstructPerfData; it can be used to free up all resources associated with the given NvpPerfData_t* object. It should be called once the user is done with the object.

```

/* Inputs
*   perfData            An NvpPerfData_t* object
*   sampleStartTimeMark 64-bit time-mark value (as returned by NvpGetTimeMark()
API);
*                           specifying the start time of the latency sample
*   sampleEndTimeMark   64-bit time-mark value (as returned by NvpGetTimeMark()
API);
*                           specifying the end time of the latency sample

```

```

/*
 * Output (NvpStatus_t)
 *   NVP_PASS           The API call completed successfully
 *   NVP_FAIL_NULLPTR   Call failed because one of the input pointers was NULL
 *   NVP_FAIL_LOGGING_STOPPED Call failed because data logging has been stopped in the
library
 *   NVP_FAIL_NO_INIT    Call failed because the given perfData object had not
been initialized
*/
NvpStatus_t
NvpRecordSample(NvpPerfData_t     *perfData,
                uint64_t          sampleStartTimeMark,
                uint64_t          sampleEndTimeMark);

/* Example Usage */
uint64_t startTimeMark, endTimeMark;

startTimeMark = NvpGetTimeMark();
/* <Instrumented-Code> */
endTimeMark = NvpGetTimeMark();

/* Assuming that perfData object has been properly initialized */
NVP_CHECKERR_EXIT(NvpRecordSample(&perfData, startTimeMark, endTimeMark));

```

NvpDumpData

This API can be used to record the data gathered (up-till the moment when this call is made) in NvpPerfData_t* object into a file in the file-system. The name of the file is taken from the object itself; as provided at the time of object initialization via NvpConstructPerfData API.

```

/* Inputs
 *   perfData           An NvpPerfData_t* object
 *
 * Output (NvpStatus_t)
 *   NVP_PASS           The API call completed successfully
 *   NVP_FAIL_NULLPTR   Call failed because the input pointer was NULL
 *   NVP_FAIL_NO_INIT    Call failed because the given perfData object had not been
initialized
*/
NvpStatus_t
NvpDumpData(NvpPerfData_t *perfData);

/* Example Usage */
NVP_CHECKERR_EXIT(NvpDumpData(&perfData));

```

NvpCalcStats

This API can be used to record the data gathered (up-till the moment when this call is made) in NvpPerfData_t* object into a file in the file-system. The name of the file is taken from the object itself; as provided at the time of object initialization via NvpConstructPerfData API.

```
/* Inputs
```

```

/*      perfData          An NvpPerfData_t* object
*      stats             An NvpPerfStats_t object in which the stats calculated for
*      unit              perfData will be recorded and reported
*      An NvpTimeUnit_t variable indicating the time-unit in which
*      the stats should be reported
*
* Output (NvpStatus_t)
*   NVP_PASS           The API call completed successfully
*   NVP_FAIL_NULLPTR  Call failed because the input pointer was NULL
*   NVP_FAIL_NO_SAMPLES Call failed because the given perfData object did not
contain
*                      any latency samples
*   NVP_FAIL_INVALID_UNIT Call failed because the specified time-unit is not
understood
*                      by the library
*/
NvpStatus_t
NvpCalcStats(NvpPerfData_t *perfData,
             NvpPerfStats_t *stats,
             NvpTimeUnits_t unit);

/* Example Usage */
NvpPerfStats_t stats;
NvpTimeUnits_t unit = USEC;

NVP_CHECKERR_EXIT(NvpCalcStats(&perfData, &stats, unit));

```

NvpPrintStats

This API can be used to print a report on the console / system logger about the statistics calculated for the given perf. data object. It can be very handy to get quick insight into the data w/o resorting to detailed offline analysis.

```

/* Inputs
*   perfData          An NvpPerfData_t* object
*   stats             An NvpPerfStats_t object containing the stats already
calculated
*                      for the given perf. data object. If this argument is NULL,
the API
*                      will first calculate stats for the perf. data by invoking
NvpCalcStats() internally
*   unit              An NvpTimeUnit_t variable indicating the time-unit in which
the stats should be reported
*   msg               A character string which should be printed alongside the
perf. report
*   csv               A boolean flag. If true, it forces the API to print the
report in CSV
*                      (comma separated values) format
*
* Output (NvpStatus_t)
*   NVP_PASS           The API call completed successfully
*   NVP_FAIL_NULLPTR  Call failed because the input pointer was NULL
*   NVP_FAIL_NO_SAMPLES Call failed because the given perfData object did not
contain

```

```

/*
 *      any latency samples
 *      NVP_FAIL_INVALID_UNIT Call failed because the specified time-unit is not
understood
 *                      by the library
 */
NvpStatus_t
NvpPrintStats(NvpPerfData_t      *perfData,
              NvpPerfStats_t    *stats,
              NvpTimeUnits_t    unit,
              char const        *msg,
              bool               csv);

/* Example Usage */
NvpTimeUnits_t unit = USEC;

NVP_CHECKERR_EXIT(NvpPrintStats(&perfData, NULL, unit, "Execution Latencies", false));

```

NvpPrintStatsExt

This is a superset of the NvpPrintStats API and provides additional arguments to specify the receiver of statical report, as well as an option to save the report to a file in target.

<pre> /* Inputs * perfData * stats calculated * the API * * * unit * * msg perf. report * csv report in CSV * * logBackend report * reportFilename which the report * file * * Output (NvpStatus_t) * NVP_PASS * NVP_FAIL_NULLPTR * NVP_FAIL_NO_SAMPLES contain * * NVP_FAIL_INVALID_UNIT understood *</pre>	<p>An NvpPerfData_t* object An NvpPerfStats_t object containing the stats already calculated for the given perf. data object. If this argument is NULL, will first calculate stats for the perf. data by invoking NvpCalcStats() internally An NvpTimeUnit_t variable indicating the time-unit in which the stats should be reported A character string which should be printed alongside the perf. report A boolean flag. If true, it forces the API to print the (comma separated values) format An enum of type NvpLogBackend_t; specifying the receiver of report A character string; specifying the name of the file in which the report should be saved. Can be NULL; to skip saving the report to file The API call completed successfully Call failed because the input pointer was NULL Call failed because the given perfData object did not contain any latency samples Call failed because the specified time-unit is not understood by the library </p>
---	---

```

*     NVP_FAIL_INVALID_LOG_BACKEND
*             The specified backend to receive the report is not
recognized
*                     by the library
*/
NvpStatus_t
NvpPrintStats(NvpPerfData_t      *perfData,
              NvpPerfStats_t   *stats,
              NvpTimeUnits_t   unit,
              char const       *msg,
              bool             csv,
              NvpLogBackend_t  logBackend,
              char const       *reportFilename);

/* Example Usage */
NvpTimeUnits_t unit = USEC;
NvpLogBackend_t backend = CONSOLE;

NVP_CHECKERR_EXIT(NvpPrintStatsExt(&perfData, NULL, unit, "Execution Latencies", false,
                                     backend, "perfSummary.txt"));

```

NvpAggregatePerfData

This API can be used to accumulate latencies stored in multiple different NvpPerfData_t objects, provided as input to the API in an array and into a single NvpPerfData_t object that is populated by the API as output. The timestamp of each sample in the aggregated data structure is taken from the timestamp of the respective sample in the first data structure in the input perf-data array (that is, `inputPerfdataArray[0]` in the arguments below). The input perf. data structures must contain the same number of samples.

```

/* Inputs
 *     netPerfData           An NvpPerfData_t* object; appropriately initialized. The API
will
*                           populate this data-structure with accumulated latencies for
each
*                           sample in the input data-structures
*     inputPerfdataArray    An array of NvpPerfData_t* pointers; containing references
to the
*                           input perf data-structures which need to be aggregated
*     numOfPerfDataObjs     Integer specifying the num. of elements in the
inputPerfdataArray
*

* Output (NvpStatus_t)
*     NVP_PASS               The API call completed successfully
*     NVP_FAIL_NULLPTR        Call failed because one of the input pointers was NULL
*     NVP_FAIL_NOINIT         Call failed because the output "netPerfData" structure is
not initialized
*     NVP_FAIL_SAMPLE_COUNT_MISMATCH
*                           Call failed because the input perf. data objects do not
contain equal
*                           number of samples
*/
NvpStatus_t
NvpAggregatePerfData(NvpPerfData_t      *netPerfData,

```

```

        NvpPerfStats_t    **inputPerfdataArray,
        uint32_t           numOfPerfDataObjs);

/* Example Usage */
#define NUM_OF_PERF_DATA_OBJS (2U)

NvpPerfData_t netPerfData, perfData1, perfData2;
NvpPerfData_t* inputPerfdataArray[NUM_OF_PERF_DATA_OBJS];

NVP_CHECKERR_EXIT(NvpConstructPerfData(&netPerfData, ...));

/* Code to initialize and gather latencies into perfData1, perfData2 */
...

inputPerfdataArray[0] = &perfData1;
inputPerfdataArray[1] = &perfData2;

NVP_CHECKERR_EXIT(NvpAggregatePerfData(&netPerfData, inputPerfdataArray,
NUM_OF_PERF_DATA_OBJS));

```

NvpRateLimitInit

This API can be used to initialize an `NvpRateLimitInfo_t*` object with the information required to enforce a desired periodicity to the target benchmark.

```

/* Inputs
 *   rtInfo          An NvpRateLimitInfo_t* object
 *   fps             32-bit variable specifying the rate-limit for the benchmark
in
 *
argument;*
*
which is taken to mean that benchmark should not be rate-
limited
*
* Output (NvpStatus_t)
*   NVP_PASS        The API call completed successfully
*   NVP_FAIL_NULLPTR Call failed because the input pointer was NULL
*/
NvpStatus_t
NvpRateLimitInit(NvpRateLimitInfo_t *rtInfo,
                  uint32_t         fps);

/* Example Usage */
uint32_t fps = 30;
NvpRateLimitInfo_t rtInfo;

NVP_CHECKERR_EXIT(NvpRateLimitInit(&rtInfo, fps));

```

NvpMarkPeriodicExecStart

This is a complementary API that must be invoked just before the benchmark is about to enter periodic execution phase. When this API is invoked, the library will internally calculate a timestamp in usec and store it as the start time of the first period of the

calling benchmark inside the given rtInfo object. This start time value is then used to calculate the boundaries for all the subsequent periods of the benchmark.

```
/* Inputs
 *   rtInfo           An NvpRateLimitInfo_t* object
 *
 * Output (NvpStatus_t)
 *   NVP_PASS         The API call completed successfully
 *   NVP_FAIL_NULLPTR Call failed because the input pointer was NULL
 */
NvpStatus_t
NvpMarkPeriodicExecStart(NvpRateLimitInfo_t *rtInfo);

/* Example Usage */
NVP_CHECKERR_EXIT(NvpMarkPeriodicExecStart(&rtInfo));
```

NvpRateLimitWait

This API can be used to wait till the start of next period during the steady-state execution of the benchmark. Internally, when this API is called, it makes the library capture a timestamp in usec and use it to calculate the time remaining till the beginning of the next period of the benchmark. If the remaining time is non-negative, then a sleep is invoked for the respective duration.

```
/* Inputs
 *   rtInfo           An NvpRateLimitInfo_t* object
 *
 * Output (NvpStatus_t)
 *   NVP_PASS         The API call completed successfully
 *   NVP_FAIL_NULLPTR Call failed because the input pointer was NULL
 */
NvpStatus_t
NvpRateLimitWait(NvpRateLimitInfo_t *rtInfo);

/* Example Usage */
NVP_CHECKERR_EXIT(NvpRateLimitWait(&rtInfo));
```

NvpCheckLibVersion

This API can be used to ensure that the library version used in the compiled benchmark code is the same as the version of the shared object file of the library runtime present on target.

```
/* Inputs
 *   None
 *
 * Output (NvpStatus_t)
 *   NVP_PASS          Version match was successful
 *   NVP_FAIL_VERSION_MISMATCH Library version does not match between benchmark
 *                               code and library runtime
 */
NvpStatus_t
NvpCheckLibVersion(void);
```

```
/* Example Usage */
NVP_CHECKERR_EXIT(NvpCheckLibVersion());
```

12.6.4 Usage Examples

The following show benchmarking library usage examples.

Enabling Data-Collection and Reporting

```
#include "nvplayfair.h"

typedef struct {
    uint32_t fps;
    uint32_t numSamples;
    ...
} testArgs_t;

int main(int argc, char **argv)
{
    NvpPerfData_t latencies;
    uint64_t startTimeMark, endTimeMark;
    testArgs_t *testArgs = parse_cli_args();

    /* Setup */
    NVP_CHECKERR_EXIT(NvpConstructPerfData(&latencies, testArgs->numSamples,
    "myLatencies.csv"));
    ...

    for (uint32_t i = 0U; i < testArgs->numSamples; ++i) {
        startTimeMark = NvpGetTimeMark();

        /* Perform required functions */
        ...

        endTimeMark = NvpGetTimeMark();
        NVP_CHECKERR_EXIT(NvpRecordSample(&latencies, startTimeMark, endTimeMark));
    }

    /* Print report */
    NVP_CHECKERR_EXIT(NvpPrintStats(&latencies, NULL, USEC, "My Test Latencies"));
    NVP_CHECKERR_EXIT(NvpDumpData(&latencies));

    /* Cleanup */
    NVP_CHECKERR_EXIT(NvpDestroyPerfData(&latencies));

    return 0;
}
```

Adding Rate-Limit to a Benchmark

```
#include "nvplayfair.h"

typedef struct {
```

```

    uint32_t fps;
    uint32_t num0fSamples;
    ...
} testArgs_t;

int main(int argc, char **argv)
{
    NvpRateLimitInfo_t rateLimitInfo;
    testArgs_t *testArgs = parse_cli_args();

    /* Setup */
    NVP_CHECKERR_EXIT(NvpRateLimitInit(&rateLimitInfo, testArgs->fps));
    ...

    NVP_CHECKERR_EXIT(NvpMarkPeriodicExecStart(&rateLimitInfo));
    for (uint32_t i = 0U; i < testArgs->num0fSamples; ++i) {
        /* Perform required functions */
        ...

        NVP_CHECKERR_EXIT(NvpRateLimitWait(&rateLimitInfo));
    }

    return 0;
}

```

12.6.5 Measuring CPU Utilization in Linux

This topic describes two methods, the `ftrace` and `top` tools, for measuring CPU utilization of workloads that run in NVIDIA DRIVE OS Linux.

Guidelines

These are the guidelines:

- > Workload should be executed for a minimum of 60 seconds when you collect utilization data.
- > It is important to collect idle system utilization for the duration of experiment and use it as a baseline for comparison.
 - Total idle utilization should be subtracted from the workload utilization for precise results.
- > For runtime CPU utilization measurements, begin the trace collection/measurement tool after the initialization is complete and stop the tracing before the application has exited.

Tool Comparison

This table summarizes the benefits and drawbacks of each method based on the tool used.

Tool	Benefits	Drawbacks
ftrace (Recommended)	<ul style="list-style-type: none"> > It provides fine-grained information of per-thread CPU utilization. 	<ul style="list-style-type: none"> > It does not provide a breakdown of the system CPU utilization. > It requires multiple steps and post-processing before a report is generated. > It cannot be used if kernel is built without debugfs (possible in production/release versions).
top	<ul style="list-style-type: none"> > It is a simple method that requires no additional scripts or post-processing. > It gives quick insight into a workload's CPU utilization. > With the right options, it can also provide reasonable thread-level data. 	<ul style="list-style-type: none"> > Single decimal point precision for %CPU column can lead to under-counting of utilization. > Ensure that the correct set of options are used.



Note: You can also consider the mpstat and tegrastats tools; however, both tools provide processor-level reports, which are not useful for fine-grained, thread-level utilization.

12.6.5.1 Using the ftrace Method

The following procedure summarizes the steps for using the ftrace method to measure CPU utilization:

1. Reset the ftrace buffers and set up the debugfs for scheduler tracing.
 2. Enable trace collection.
 3. Run the workload.
-
- Note:** See the [Guidelines](#) section for considerations regarding running the workload.
4. Stop tracing and save the trace data.
 5. Collect another report for an idle system for the duration of experiment but without workload, for example, sleep 60.

All of the preceding steps are automated in the following ftrace script. Use this entire script for measuring CPU utilization.

```
#!/bin/bash

INIT_TIME=10
EXIT_TIME=10
TEST_TIME=60
APP_RUN_TIME=$((INIT_TIME + TEST_TIME + EXIT_TIME))

# >>>>>>>>>>>> FIXME <<<<<<<<<<
# SPECIFY THE FILENAME FOR STORING TRACE DATA

IDLE_TRACE_FILE=<MY-IDLE-TRACE>.txt
WORKLOAD_TRACE_FILE=<MY-WORKLOAD-TRACE>.txt

# >>>>>>>>>>>> FIXME <<<<<<<<<
# POPULATE THE FOLLOWING FUNCTION AS PER YOUR WORKLOAD

runWorkload () {
    # Sepcify the command-line to execute the workload.
    # Make sure that the workload executes for at-least
    # ${APP_RUN_TIME} seconds.

    echo "[FG-SHELL] Running workload..."
    ./<MY COMMAND-LINE FOR RUNNING WORKLOAD>

    echo "[FG-SHELL] Workload complete!"
}

#####
#      DO NOT CHANGE ANYTHING IN THE SCRIPT BELOW      #
#####

shopt -s expand_aliases

tracefs="/sys/kernel/debug/tracing"
schedTraceEvents="$tracefs/events/sched"

alias displayTraceBuffer='cat ${tracefs}/trace'
alias flushTraceBuffer='echo > ${tracefs}/trace'
alias turnTracingOn='echo 1 > ${tracefs}/tracing_on'
alias turnTracingOff='echo 0 > ${tracefs}/tracing_on'
alias ownTracefsDir='sudo chown -R nvidia:nvidia ${tracefs}'
alias makeTraceClockMonotonic='echo mono > ${tracefs}/trace_clock'
alias increasePerCoreTraceBufferSize='echo 10000 > ${tracefs}/buffer_size_kb'
alias enableSchedTraceEvents='echo 1 > ${schedTraceEvents}/sched_switch/enable;
                           echo 1 > ${schedTraceEvents}/sched_wakeup/enable;
                           echo 1 > ${schedTraceEvents}/sched_wakeup_new/enable'

configureFtrace () {
    echo "[FG-SHELL] Configuring ftrace..."

    ownTracefsDir
    turnTracingOff
```

```

makeTraceClockMonotonic
increasePerCoreTraceBufferSize
enableSchedTraceEvents
}

collectIdleSystemTrace () {
    echo "[FG-SHELL] Collecting idle system trace for ${TEST_TIME} sec..."
    flushTraceBuffer
    turnTracingOn

    sleep ${TEST_TIME}
    turnTracingOff

    echo "[FG-SHELL] Saving idle system trace..."
    displayTraceBuffer &> ${IDLE_TRACE_FILE}
    flushTraceBuffer

    echo "[FG-SHELL] Idle system trace can be seen here: ${IDLE_TRACE_FILE}"
}

configureFtrace
collectIdleSystemTrace

# Invoke background sub-shell to collect trace with workload
(
    echo "[BG-SHELL] Sleeping for init time: ${INIT_TIME} sec..."
    sleep ${INIT_TIME}

    echo "[BG-SHELL] Starting trace-collection..."
    turnTracingOn

    sleep ${TEST_TIME}
    turnTracingOff

    echo "[BG-SHELL] Tracing complete. Recording data..."
    displayTraceBuffer &> ${WORKLOAD_TRACE_FILE}

    echo "[BG-SHELL] Workload trace data can be seen here: ${WORKLOAD_TRACE_FILE}"
) &

# Execute workload in the foreground shell
runWorkload

```

Running the above script generates the following output on console:

```
[FG-SHELL] Configuring ftrace...
[FG-SHELL] Collecting idle system trace for 60 sec...
[FG-SHELL] Saving idle system trace...
[FG-SHELL] Idle system trace can be seen here: idle_trace.txt
[FG-SHELL] Running workload...
[BG-SHELL] Sleeping for init time: 10 sec...
[BG-SHELL] Starting trace-collection...
[BG-SHELL] Tracing complete. Recording data...
[BG-SHELL] Workload trace data can be seen here: workload_trace.txt
[FG-SHELL] Workload complete!
```



Note: The steps shown in the output appear in the exact same order as the script.

- Analyze generated traces (idle system with workload) to create precise utilization reports by using the following command:

```
./cpuUtilMain.py -o linux -t <file-containing-workload-ftrace-data> [-i <file-containing-idle-system-ftrace-data>]
```



Note: You can find cpuUtilMain.py in DRIVE Linux SDK in the <sdk-install-dir>/drive-linux/tools/nvplayfair/cpu_util/ directory.

The post-processing script will generate CPU utilization summary on console and save the detailed per-thread utilization report to a file.

```

waqara@build-waqara-20220628T103507008:~/automotive-dev-main/

INFO: Analyzing workload trace data for Linux
INFO: Progress - 100.0% | Line # - 186252 / 186252
INFO: Per-Thread CPU utilization data saved to file: ./si

INFO: Generating report

----- CPU Utilization Report -----

Trace Duration : 86.526 seconds
Total System Utilization : 5.437%

***** Top-10 Threads *****

# Thread UI
-----
1 SIPL_ICP_ISP_0 32
2 Main 32
3 irq/203-b950000 10
4 irq/174-host_sy 27
5 PipelineEvent 32
6 nvgpu_channel_p 90
7 kworker/u22:3 24
8 capture 24
9 DEVBLK_WORKER_0 32
10 FrameQueue 32

```

12.6.5.2 Using the top Method

The following procedure summarizes the steps for using the top method:

1. Run the top tool with the specified options to measure the CPU utilization.

For example,

```
#!/bin/bash

# -b (batch mode): useful for sending top output to a file. Allows top to execute for
# the given iteration count and then exit
# -d <duration>: run for <duration> seconds
# -n 2: capture two snapshots (in the given duration)
# -H: collect thread level stats.
# -i: hide idle processes
# -w 200: Set line width to 200; to get non-truncated thread names

top -b -i -d <duration> -n 2 -H -w 200 &> top_output.txt &
```

- Run the workload.



Note: See the [Guidelines](#) section for considerations regarding running the workload.

- Collect another report for an idle system using the same top command.
- View the files containing the output of the top command to see the CPU utilization for the idle system and with workload.
 - To determine the overall CPU utilization from the top output, use the **second** occurrence of the line that starts with **%Cpu(s)** as follows:

```
top - 23:51:36 up 1:25, 3 users, load average: 0.22, 0.31, 0.25
Threads: 511 total, 4 running, 507 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.9 us, 8.7 sy, 0.0 ni, 89.4 id, 0.0 wa, 0.0 hi, 0.0 si,
MiB Mem : 28380.7 total, 26130.6 free, 1726.3 used, 523.8 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 26295.2 avail Mem

          PID USER      PR  NI      VIRT      RES      SHR S %CPU %MEM     TIME+ CO
        3425 root      -11   0    24952   14124    5736 R 99.9  0.0  0:00.17 Ma
        3423 nvidia     20   0     8128    3488    2588 R 11.8  0.0  0:00.03 to

top - 23:52:56 up 1:26, 3 users, load average: 0.06, 0.24, 0.23
Threads: 519 total, 1 running, 518 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.4 us, 0.1 sy, 0.0 ni, 99.6 id, 0.0 wa, 0.0 hi, 0.0 si,
MiB Mem : 28380.7 total, 26128.1 free, 1729.0 used, 523.6 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 26292.7 avail Mem
```

- The following three values are the important numbers for measuring CPU utilization:
 - > **%Cpu(s): 0.4 us, 0.1 sy, 0.0 ni, 99.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st**
 - us – %User time, sy – %Sys time, id – %Idle time
 - > Total CPU utilization to be derived from %idle time by subtracting it from 100. This is the percentage of the total system.

- > Get the CPU utilization in terms of percentage of cores by multiplying the preceding value with the number of cores in the system. For example,

```
CPU util: (100 - 99.6) * 11 = 4.4% (assuming 11 cores in the system)
```

Chapter 13. Manifest

After installation of foundation, these directories are created:

- > NVIDIA DRIVE® Foundation:

```
<top>/drive-foundation
```

- > The toolchains:

```
<top>/toolchains
```

NVIDIA DRIVE Foundation Directory

The drive-foundation directory components are as follows.

Directory	Description
firmwares/	Holds the firmware.
hypervisor/	Holds the hypervisor configurations.
platform-config/	Holds files that support platform configuration.
tools/	Holds the DRIVE OTA, EMC and host tools.
utils/	Hold the utility scripts.
virtualization/	Holds the build-configs, hypervisor, pct, services, tools, virt and vm-server directories that support virtualization.

Toolchains Directory

Consult the NVIDIA Supported Cross Toolchains chapter under System Programming in the *NVIDIA DRIVE OS 6.0 Linux PDK Developer Guide*.



Note: The *NVIDIA DRIVE OS Linux PDK Developer Guide* requires specific agreements with NVIDIA. Consult with your NVIDIA Customer Support Engineer for more information.

13.1 Linux SDK

After installation, these directories are created:

- > The NVIDIA DRIVE® directory, ./drive-linux
- > The CUDA directory, ./drive-cuda
- > The toolchains directory, ./toolchains
- > The foundation directory, ./drive-foundation
- > The hardware-specific directory, ./hardware (PDK only)

This topic describes the contents of the NVIDIA DRIVE directory and the CUDA directory. The toolchains directory and the foundation directory are part of NVIDIA DRIVE Foundation 6.0SDK, and are described in the *NVIDIA DRIVE Foundation SDK Development Guide*.

13.1.1 Manifest

After installation of foundation, these directories are created:

- > NVIDIA DRIVE® Foundation:
 <top>/drive-foundation
- > The toolchains:
 <top>/toolchains

NVIDIA DRIVE Foundation Directory

The drive-foundation directory components are as follows.

Directory	Description
firmwares/	Holds the firmware.
hypervisor/	Holds the hypervisor configurations.
platform-config/	Holds files that support platform configuration.
tools/	Holds the DRIVE OTA, EMC and host tools.
utils/	Hold the utility scripts.
virtualization/	Holds the build-configs, hypervisor, pct, services, tools, virt and vm-server directories that support virtualization.

Toolchains Directory

Consult the NVIDIA Supported Cross Toolchains chapter under System Programming in the *NVIDIA DRIVE OS 6.0 Linux PDK Developer Guide*.



Note: The *NVIDIA DRIVE OS Linux PDK Developer Guide* requires specific agreements with NVIDIA. Consult with your NVIDIA Customer Support Engineer for more information.

13.1.2 CUDA Directory

The following table lists the components of the `drive-cuda` directory.

Folder	Description
<code>target/</code>	Contains the <i>target</i> CUDA Debian installer package for installing on the target.

Chapter 14. Device Tree

The device tree is used for storing platform-specific configuration for system resources used by DRIVE OS modules.

14.1 Display Device Tree

The NvDisplay driver uses the following device tree nodes and properties. Properties customizable are marked accordingly for each property used. Nodes used in NvDisplay drivers for QNX are as follows.

NvDisplay

nvdisplay node, contains configuration parameters for initializing the NvDisplay driver.

```
nvdisplay: display@13800000 {
    compatible = "nvidia,tegra234-display";
    power-domains = <&bpmp TEGRA234_POWER_DOMAIN_DISP>; nvidia,num-dpaux-instance = <1>;
    reg-names = "nvdisplay", "dpaux0", "hdacodec", "mipical"; reg = <0x0 0x13800000 0x0
        0xFFFF /* nvdisplay */
    0x0 0x155C0000 0x0 0xFFFF /* dpaux0 */ 0x0 0x0242c000 0x0 0x1000 /* hdacodec */ 0x0
        0x03990000 0x0 0x1000>; /* mipical */
    interrupt-names = "nvdisplay", "dpaux0", "hdacodec"; interrupts = <0 416 4
    0 419 4
    0 61 4>;
    nvidia,bpmp = <&bpmp>;
    clocks = <&bpmp_clks TEGRA234_CLK_HUB>,
    <&bpmp_clks TEGRA234_CLK_DISP>,
    <&bpmp_clks TEGRA234_CLK_NVDISPLAY_P0>,
    <&bpmp_clks TEGRA234_CLK_NVDISPLAY_P1>,
    <&bpmp_clks TEGRA234_CLK_DPAUX>,
    <&bpmp_clks TEGRA234_CLK_FUSE>,
    <&bpmp_clks TEGRA234_CLK_DSIPLL_VCO>,
    <&bpmp_clks TEGRA234_CLK_DSIPLL_CLKOUTPN>,
    <&bpmp_clks TEGRA234_CLK_DSIPLL_CLKOUTA>,
    <&bpmp_clks TEGRA234_CLK_SPPLL0_VCO>,
    <&bpmp_clks TEGRA234_CLK_SPPLL0_CLKOUTPN>,
    <&bpmp_clks TEGRA234_CLK_SPPLL0_CLKOUTA>,
    <&bpmp_clks TEGRA234_CLK_SPPLL0_CLKOUTB>,
    <&bpmp_clks TEGRA234_CLK_SPPLL0_DIV10>,
    <&bpmp_clks TEGRA234_CLK_SPPLL0_DIV25>,
    <&bpmp_clks TEGRA234_CLK_SPPLL0_DIV27PN>,
```

```

<&bpmp_clks TEGRA234_CLK_SPPLL1_VCO>,
<&bpmp_clks TEGRA234_CLK_SPPLL1_CLKOUTPN>,
<&bpmp_clks TEGRA234_CLK_SPPLL1_DIV27PN>,
<&bpmp_clks TEGRA234_CLK_VPLL0_REF>,
<&bpmp_clks TEGRA234_CLK_VPLL0>,
<&bpmp_clks TEGRA234_CLK_VPLL1>,
<&bpmp_clks TEGRA234_CLK_NVDISPLAY_P0_REF>,
<&bpmp_clks TEGRA234_CLK_RG0>,
<&bpmp_clks TEGRA234_CLK_RG1>,
<&bpmp_clks TEGRA234_CLK_DISPPLL>,
<&bpmp_clks TEGRA234_CLK_DISPHUBPLL>,
<&bpmp_clks TEGRA234_CLK_DSI_LP>,
<&bpmp_clks TEGRA234_CLK_DSI_CORE>,
<&bpmp_clks TEGRA234_CLK_DSI_PIXEL>,
<&bpmp_clks TEGRA234_CLK_PRE_SOR0>,
<&bpmp_clks TEGRA234_CLK_PRE_SOR1>,
<&bpmp_clks TEGRA234_CLK_DP_LINK_REF>,
<&bpmp_clks TEGRA234_CLK_SOR_LINKA_INPUT>,
<&bpmp_clks TEGRA234_CLK_SOR_LINKA_AFIFO>,
<&bpmp_clks TEGRA234_CLK_SOR_LINKA_AFIFO_M>,
<&bpmp_clks TEGRA234_CLK_RG0_M>,
<&bpmp_clks TEGRA234_CLK_RG1_M>,
<&bpmp_clks TEGRA234_CLK_SOR0_M>,
<&bpmp_clks TEGRA234_CLK_SOR1_M>,
<&bpmp_clks TEGRA234_CLK_PLLHUB>,
<&bpmp_clks TEGRA234_CLK_SOR0>,
<&bpmp_clks TEGRA234_CLK_SOR1>,
<&bpmp_clks TEGRA234_CLK_SOR_PAD_INPUT>,
<&bpmp_clks TEGRA234_CLK_PRE_SF0>,
<&bpmp_clks TEGRA234_CLK_SF0>,
<&bpmp_clks TEGRA234_CLK_SF1>,
<&bpmp_clks TEGRA234_CLK_DSI_PAD_INPUT>,
<&bpmp_clks TEGRA234_CLK_PRE_SOR0_REF>,
<&bpmp_clks TEGRA234_CLK_PRE_SOR1_REF>,
<&bpmp_clks TEGRA234_CLK_SOR0_PLL_REF>,
<&bpmp_clks TEGRA234_CLK_SOR1_PLL_REF>,
<&bpmp_clks TEGRA234_CLK_SOR0_REF>,
<&bpmp_clks TEGRA234_CLK_SOR1_REF>,
<&bpmp_clks TEGRA234_CLK_OSC>,
<&bpmp_clks TEGRA234_CLK_DSC>,
<&bpmp_clks TEGRA234_CLK_MAUD>,
<&bpmp_clks TEGRA234_CLK_AZA_2XBIT>,
<&bpmp_clks TEGRA234_CLK_AZA_BIT>,
<&bpmp_clks TEGRA234_CLK_MIPI_CAL>,
<&bpmp_clks TEGRA234_CLK_UART_FST_MIPI_CAL>,
<&bpmp_clks TEGRA234_CLK_SOR0_DIV>; clock-names = "nvdisplayhub_clk",
"nvdisplay_disp_clk", "nvdisplay_p0_clk", "nvdisplay_p1_clk", "dpaux0_clk", "fuse_clk",
"dsipll_vco_clk", "dsipll_clkoutpn_clk", "dsipll_clkouta_clk", "sppll0_vco_clk",
"sppll0_clkoutpn_clk", "sppll0_clkouta_clk", "sppll0_clkoutb_clk", "sppll0_div10_clk",
"sppll0_div25_clk", "sppll0_div27_clk",
"sppll1_vco_clk", "sppll1_clkoutpn_clk", "sppll1_div27_clk", "vp1l0_ref_clk",
"vp1l0_clk", "vp1l1_clk",
"nvdisplay_p0_ref_clk", "rg0_clk",

```

```

"rg1_clk", "disppll_clk", "disphubpll_clk", "dsi_lp_clk", "dsi_core_clk",
"dsi_pixel_clk", "pre_sor0_clk", "pre_sor1_clk", "dp_link_ref_clk",
"sor_linka_input_clk", "sor_linka_afifo_clk",
"sor_linka_afifo_m_clk", "rg0_m_clk", "rg1_m_clk", "sor0_m_clk", "sor1_m_clk",
"pllhub_clk",
"sor0_clk", "sor1_clk", "sor_pad_input_clk", "pre_sf0_clk", "sf0_clk",
"sf1_clk", "dsi_pad_input_clk", "pre_sor0_ref_clk", "pre_sor1_ref_clk",
"sor0_ref_pll_clk", "sor1_ref_pll_clk", "sor0_ref_clk", "sor1_ref_clk", "osc_clk",
"dsc_clk", "maud_clk", "aza_2xbit_clk", "aza_bit_clk", "mipi_cal_clk",
"uart_fst_mipi_cal_clk", "sor0_div_clk";
resets = <&bpmp_resets TEGRA234_RESET_NVDISPLAY>,
<&bpmp_resets TEGRA234_RESET_DPAUX>,
<&bpmp_resets TEGRA234_RESET_DSI_CORE>,
<&bpmp_resets TEGRA234_RESET_MIPI_CAL>;
reset-names = "nvdisplay_reset",
"dpaux0_reset", "dsi_core_reset", "mipi_cal_reset";
status = "disabled";
nvidia,disp-sw-soc-chip-id = <0x2350>;
#if TEGRA_IOMMU_DT_VERSION >= DT_VERSION_2
interconnects = <&mc TEGRA234_MEMORY_CLIENT_NVDISPLAYR>,
<&mc TEGRA234_MEMORY_CLIENT_NVDISPLAYR1>;
interconnect-names = "dma-mem", "read-1";
#endif
iommu = <&smmu_iso TEGRA_SID_ISO_NVDISPLAY>; non-coherent;
nvdisplay-niso {
compatible = "nvidia,tegra234-display-niso";
iommu = <&smmu_niso0 TEGRA_SID_NISO0_NVDISPLAY>; dma-coherent;
};
dsi {
compatible = "nvidia,tegra234-dsi"; nvidia,active-panel = "NULL"; status = "disabled";
};
};

```

compatible:

Description: - `compatible` contains the unique string to identify the external NvDisplay DT node.

Customizable: No

Optional: No

Value: " nvidia,tegra234-display"

power-domains:

Description: - `power-domains` identifies the power domain display belongs to.

Customizable: No

Optional: No

Value: <&bpmp TEGRA234_POWER_DOMAIN_DISP>

nvidia,num-dpaux-instance:

Description: - num-dpaux-instance specifies the number of DPAUX pads on the underlying platform.

Customizable: No

Optional: No

Value: <1>

reg-names:

Description: - reg-names mentions all of the MMIO device apertures that the display driver needs access to.

Customizable: No

Optional: No

Value: Must contain an entry for all the register names

- > nvdisplay
- > dpaux0
- > hd(codec (Not used by QNX driver))
- > mipical (Not used by QNX driver)

reg:

Description: - Physical base address and length of the controller's registers

Customizable: No

Optional: No

Value: Tuple in the form of address and length/size that describes the address range. Must contain an entry for each register entry mentioned in the "reg" field:

- > 0x0 0x13800000 0x0 0xFFFF
- > 0x0 0x155C0000 0x0 0xFFFF
- > 0x0 0x0242c000 0x0 0x1000 (corresponds to hd(codec, Not used by QNX driver))
- > 0x0 0x03990000 0x0 0x10000 (corresponds to mipical, Not used by QNX driver)

interrupt-names:

Description: - Mentions the type of interrupts supported by NvDisplay node

Customizable: No

Optional: No

Value: Must contain the following entries:

- > nvdisplay

- > dpaux0
- > hd(codec (Not used by QNX driver))

interrupts:

Description: `interrupts` holds the IRQ number and IRQ type connected to NvDisplay.

Customizable: No

Optional: No

Value: Must contain an entry for each entry in interrupt-names.

0 416 4

0 419 4

0 61 4

nvidia,bpmp:

Description: BPMP device, needed for NvDisplay

Customizable: No

Optional: No

Value: bpmp

clocks:

Description: `clocks` holds the type of clock IDs supported by NvDisplay.

Customizable: No

Optional: No

Value: <&bpmp_clks TEGRA234_CLK_HUB>,
<&bpmp_clks TEGRA234_CLK_DISP>,
<&bpmp_clks TEGRA234_CLK_NVDISPLAY_P0>,
<&bpmp_clks TEGRA234_CLK_NVDISPLAY_P1>,
<&bpmp_clks TEGRA234_CLK_DPAUX>,
<&bpmp_clks TEGRA234_CLK_FUSE>,
<&bpmp_clks TEGRA234_CLK_DSIPLL_VCO>,
<&bpmp_clks TEGRA234_CLK_DSIPLL_CLKOUTPN>,
<&bpmp_clks TEGRA234_CLK_DSIPLL_CLKOUTA>,

```
<&bpmp_clks TEGRA234_CLK_SPPLL0_VCO>,
<&bpmp_clks TEGRA234_CLK_SPPLL0_CLKOUTPN>,
<&bpmp_clks TEGRA234_CLK_SPPLL0_CLKOUTA>,
<&bpmp_clks TEGRA234_CLK_SPPLL0_CLKOUTB>,
<&bpmp_clks TEGRA234_CLK_SPPLL0_DIV10>,
<&bpmp_clks TEGRA234_CLK_SPPLL0_DIV25>,
<&bpmp_clks TEGRA234_CLK_SPPLL0_DIV27PN>,
<&bpmp_clks TEGRA234_CLK_SPPLL1_VCO>,
<&bpmp_clks TEGRA234_CLK_SPPLL1_CLKOUTPN>,
<&bpmp_clks TEGRA234_CLK_SPPLL1_DIV27PN>,
<&bpmp_clks TEGRA234_CLK_VPLL0_REF>,
<&bpmp_clks TEGRA234_CLK_VPLL0>,
<&bpmp_clks TEGRA234_CLK_VPLL1>,
<&bpmp_clks TEGRA234_CLK_NVDISPLAY_P0_REF>,
<&bpmp_clks TEGRA234_CLK_RG0>,
<&bpmp_clks TEGRA234_CLK_RG1>,
<&bpmp_clks TEGRA234_CLK_DISPPLL>,
<&bpmp_clks TEGRA234_CLK_DISPHUBPLL>,
<&bpmp_clks TEGRA234_CLK_DSI_LP>,
<&bpmp_clks TEGRA234_CLK_DSI_CORE>,
<&bpmp_clks TEGRA234_CLK_DSI_PIXEL>,
<&bpmp_clks TEGRA234_CLK_PRE_SOR0>,
<&bpmp_clks TEGRA234_CLK_PRE_SOR1>,
<&bpmp_clks TEGRA234_CLK_DP_LINK_REF>,
<&bpmp_clks TEGRA234_CLK_SOR_LINKA_INPUT>,
<&bpmp_clks TEGRA234_CLK_SOR_LINKA_AFIFO>,
<&bpmp_clks TEGRA234_CLK_SOR_LINKA_AFIFO_M>,
<&bpmp_clks TEGRA234_CLK_RG0_M>,
```

```
<&bpmp_clks TEGRA234_CLK_RG1_M>,
<&bpmp_clks TEGRA234_CLK_SOR0_M>,
<&bpmp_clks TEGRA234_CLK_SOR1_M>,
<&bpmp_clks TEGRA234_CLK_PLLHUB>,
<&bpmp_clks TEGRA234_CLK_SOR0>,
<&bpmp_clks TEGRA234_CLK_SOR1>,
<&bpmp_clks TEGRA234_CLK_SOR_PAD_INPUT>,
<&bpmp_clks TEGRA234_CLK_PRE_SF0>,
<&bpmp_clks TEGRA234_CLK_SF0>,
<&bpmp_clks TEGRA234_CLK_SF1>,
<&bpmp_clks TEGRA234_CLK_DSI_PAD_INPUT>,
<&bpmp_clks TEGRA234_CLK_PRE_SOR0_REF>,
<&bpmp_clks TEGRA234_CLK_PRE_SOR1_REF>,
<&bpmp_clks TEGRA234_CLK_SOR0_PLL_REF>,
<&bpmp_clks TEGRA234_CLK_SOR1_PLL_REF>,
<&bpmp_clks TEGRA234_CLK_SOR0_REF>,
<&bpmp_clks TEGRA234_CLK_SOR1_REF>,
<&bpmp_clks TEGRA234_CLK_OSC>,
<&bpmp_clks TEGRA234_CLK_DSC>,
<&bpmp_clks TEGRA234_CLK_MAUD>,
<&bpmp_clks TEGRA234_CLK_AZA_2XBIT>,
<&bpmp_clks TEGRA234_CLK_AZA_BIT>,
<&bpmp_clks TEGRA234_CLK_MIPI_CAL>,
<&bpmp_clks TEGRA234_CLK_UART_FST_MIPI_CAL>,
<&bpmp_clks TEGRA234_CLK_SOR0_DIV>;
```

clock-names:

Description: List of clock input name strings sorted in the same order as the clocks property. Consumer drivers use clock-names to match clock input names with clocks specifiers

Customizable: No

Optional: No

Value: "nvdisplayhub_clk", "nvdisplay_disp_clk", "nvdisplay_p0_clk", "nvdisplay_p1_clk", "dpaux0_clk", "fuse_clk", "dsipll_vco_clk", "dsipll_clkoutpn_clk", "dsipll_clkouta_clk", "sppll0_vco_clk", "sppll0_clkoutpn_clk", "sppll0_clkouta_clk", "sppll0_clkoutb_clk", "sppll0_div10_clk", "sppll0_div25_clk", "sppll0_div27_clk", "sppll1_vco_clk", "sppll1_clkoutpn_clk", "sppll1_div27_clk", "vppll0_ref_clk", "vppll0_clk", "vppll1_clk", "nvdisplay_p0_ref_clk", "rg0_clk", "rg1_clk", "disppll_clk", "disphubpll_clk", "dsi_lp_clk", "dsi_core_clk", "dsi_pixel_clk", "pre_sor0_clk", "pre_sor1_clk", "dp_link_ref_clk", "sor_linka_input_clk", "sor_linka_afifo_clk", "sor_linka_afifo_m_clk", "rg0_m_clk", "rg1_m_clk", "sor0_m_clk", "sor1_m_clk", "pllhub_clk", "sor0_clk", "sor1_clk", "sor_pad_input_clk", "pre_sf0_clk", "sf0_clk", "sf1_clk", "dsi_pad_input_clk", "pre_sor0_ref_clk", "pre_sor1_ref_clk", "sor0_ref_pll_clk", "sor1_ref_pll_clk", "sor0_ref_clk", "sor1_ref_clk", "osc_clk", "dsc_clk", "maud_clk", "aza_2xbit_clk", "aza_bit_clk", "mipi_cal_clk", "uart_fst_mipi_cal_clk", "sor0_div_clk";

resets:

Description: List of phandle and reset specifier pairs, one pair for each reset signal that affects the device, or that the device manages.

Customizable: No

Optional: No

Value:

- > TEGRA234_RESET_NVDISPLAY
- > TEGRA234_RESET_DPAUX
- > TEGRA234_RESET_DSI_CORE (Not used by QNX driver)
- > TEGRA234_RESET_MIPI_CAL (Not used by QNX driver)

reset-names:

Description: The name of the resets mentioned in the “resets” field

Customizable: No

Optional: No

Value: Must contain an entry for each value in the “resets” field:

- > nvdisplay_reset
- > dpaux0_reset
- > dsi_core_reset (Not used by QNX driver)
- > mipi_cal_reset (Not used by QNX driver)

status:

Description: Holds the status of NvDisplay

Customizable: No

Optional: No

Value: “okay” or “disabled”

nvidia,disp-sw-soc-chip-id:

Description: Mentions the SOC SW Chip ID

Customizable: No

Optional: No

Value: 0x2350

interconnects:

Description: Specifies the bandwidth of the clients used by NvDisplay. This is only if TEGRA_IOMMU_DT_VERSION is greater or equal to DT_VERSION_2. Not used on AV+L or AV+Q.

Customizable: No

Optional: No

Value:

- > TEGRA234_MEMORY_CLIENT_NVDISPLAYR
- > TEGRA234_MEMORY_CLIENT_NVDISPLAYR1

interconnect-names:

Description: Mentions the names of the interconnects used by NvDisplay. This is only if TEGRA_IOMMU_DT_VERSION is greater or equal to DT_VERSION_2.

Customizable: No

Optional: No

Value: Must contain an entry for each value in “interconnects” field

- > dma-mem
- > read-1

iommus:

Description: Mentions the name of the IOMMU used by NvDisplay for ISO traffic.

Customizable: No

Optional: No

Value: TEGRA_SID_ISO_NVDISPLAY

non-coherent:

Description: Present because NvDisplay ISO accesses to DRAM are non-coherent.

Customizable: No

Optional: No

Value: N/A

nvdisplay-niso node

Contains details about the IOMMU instance used by NvDisplay for NISO memory accesses.

```
nvdisplay-niso {
    compatible = "nvidia,tegra234-display-niso";
    iommus = <&smmu_niso0 TEGRA_SID_NIS00_NVDISPLAY>; dma-coherent;
};
```

compatible:

Description: - `compatible` contains the unique string to identify the external NvDisplay-NISO DT node

Customizable: No

Optional: No

Value: " nvidia,tegra234-display-niso"

iommus:

Description: Mentions the name of the IOMMU used by the NvDisplay-NISO node.

Customizable: No

Optional: No

Value:

- > TEGRA_SID_NIS00_NVDISPLAY

dma-coherent:

Description: Present because NvDisplay NISO memory accesses are DMA-coherent.

Customizable: No

Optional: No

Value: N/A

dsi node

Contains details about DSI, used by NvDisplay node. Is not used in automotive QNX.

```
dsi {
    compatible = "nvidia,tegra234-dsi"; nvidia,active-panel = "NULL"; status = "disabled";
```

```
};
```

compatible:

Description: - `compatible` contains the unique string to identify the external NvDisplay-DSI DT node.

Customizable: No

Optional: No

Value: " nvidia,tegra234-dsi"

nvidia,active-panel:

Description: - Contains the active panels under DSI

Customizable: No

Optional: No

Value: " NULL"

status:

Description: Holds the status of NvDisplay DSI node.

Customizable: No

Optional: No

Value: "okay" or "disabled"

Frozen Frame Detection

In order to enable frozen frame detection, these are the properties required. It uses the regional CRC approach.

```
regional-crc {
head0 {
/* num regions */ num-regions = <8>;

/* Array specifying "num-regions" region details.
* Each region has 4 entries in array:
* x-coordinate, y-coordinate, width and height
*/
regions = <100 100 200 200
400 300 200 200
700 100 100 300
900 200 400 300
1400 100 400 300
200 600 400 200
700 600 200 300
1200 600 400 300>;
```

```

/*
 * Display driver will use this value to declare
 * Frozen frame if CRC value for any region is repeated
 * these many times.
 */
ff-detection-threshold = <4>;
};
};

```

regional-crc:

Used to specify if regional CRC functionality can be enabled. If this entry is removed then FF detection functionality is disabled. This node is optional and only required if we need to enable Frozen Frame Detection.

head0:

Specifies which head the regional CRC is enabled on, ex:head0, head1. Orin has only two valid node names - head0 and head1. Either node, or both can be present based on which head/stream the Frozen Frame detection is to be enabled for.

num-regions:

Description: Specifies how many regions in given view-port will be used for regional crc comparison.

Customizable: Yes

Optional: No

Value: 1-9

regions:

Description: Used to mention different areas (upto 9) within a viewport on display image. Each region is described with 4 values mentioning the top left and bottom right coordinates.

Customizable: Yes

Optional: No

Value: All values are separated by space and start with values corresponding to the first region upto max allowed regions as mentioned in "num-regions" property. The position and size of each region can be specified by the user.

For example:

> <100100 200 200

400300 200 200

700100 100 300

900200 400 300

```
1400100 400 300
200600 400 200
700600 200 300
1200600 400 300>
```

ff-detection-threshold:

Description: Used to specify after how many frames the FF detection is considered to be true. If the CRC for any single region remains the same for this many frames, the display driver will treat this as an error since it indicates that at least one region of the frame is stuck.

Customizable: Yes

Optional: No

Value: Integer specifying the number of frames. For example: <4>

Drive Setmode

When driver setmode is enabled, display setup + modeset will happen during the resmgr init phase instead of deferring this to when the first display client comes up. So, enabling driver setmode can improve "power-on to first pixel visible on screen" latency. On QNX safety, driver setmode is enabled by default. On QNX Standard, driver setmode can be enabled using Device Tree. Driver setmode can be enabled only for the configuration which uses DP Serializer, on other configurations enabling it will fail to load the display driver.

```
display@13800000 { nvidia,driver-setmode;
};
```

nvidia,driver-setmode:

Description: Used to enable driver setmode.

Customizable: No

Optional: Yes

Value: N/A

Head to Window Assignment

Used to configure head to window assignment in the Device Tree. If an assignment is not specified in the DT, the driver assigns windows (2N) and (2N + 1) to HEAD N. In DT, specify the assignment using 64 bit mask, which is interpreted as:

Head-Bitmask	Window-Number
BITMASK(0-7)	0

Head-Bitmask	Window-Number
BITMASK(8-15)	1
BITMASK(16-23)	2
BITMASK(24-31)	3
BITMASK(32-39)	4
BITMASK(40-47)	5
BITMASK(48-55)	6
BITMASK(56-63)	7

The display driver fails to load if an invalid assignment is specified in the DT. The specified assignment must adhere to the conditions below:

1. The specified window number must be supported by hardware.
2. The specified head number must be supported by hardware.
3. The same window must not be assigned simultaneously for multiple heads.
4. At least one window must be assigned to at least one head (that is, the specified window-head mask should not be 0).

The display driver culls the head with no windows assigned, and all the heads above it. For example, if hardware supports three heads and the user uses the assignment mask to assign valid windows to head-0 and head-2 but no windows to head-1, then head-1 and head-2 is culled.

```
display@13800000 {
    nvidia,window-head-mask = <0x00000000 0x02010101>;
};
```

nvidia,window-head-mask:

Description: Used to specify the 64-bit window head assignment mask.

Customizable: Yes

Optional: Yes

Value: If no value is specified, a default head<->window assignment is used. If not, any value which fits the criteria mentioned above will work.

Static IMP

Starting from 6.0.5.0, QNX customers should enable static IMP. For enabling static IMP, we need to program certain IMP settings in BCT and Device Tree files. Generated using a host side tool called "laptsa-imp", we are able to generate this DT fragment.

```
static-imp-data {
/* Window and cursor pool config */
```

```
window-pool-config = <0x227 0x227 0x227 0x227>; cursor-pool-config = <0x1f 0x1f>;  
/* Window and cursor drain meter config */  
window-drain-meter-config = <0x20 0x20 0x20 0x20>; cursor-drain-meter-config = <0x3  
0x3>;  
/* Window and cursor fetch meter config */ window-fetch-meter-config = <0xf 0xf 0xf  
0xf>; cursor-fetch-meter-config = <0x2 0x2>;  
/* Delay before a START_FETCH command is sent to IsoHub */ start-fetch-delay-us = <0x1a8  
0x1a8>;  
/* Elv start value */ elv-start = <0x4 0x4>;  
/* Clock Frequencies */ hub-clock-khz = <82300>;  
disp-clock-khz = <311862>;  
};  
};
```

window-pool-config

Description: Specifies the window pool config.

Customizable: Yes

Optional: No

Value: Ex: <0x227 0x227 0x227 0x227>

cursor-pool-config

Description: Specifies the cursor pool config.

Customizable: Yes

Optional: No

Value: Ex: <0x1f 0x1f>

window-drain-meter-config

Description: Specifies the window drain meter config.

Customizable: Yes

Optional: No

Value: For example: <0x20 0x20 0x20 0x20>

cursor-drain-meter-config

Description: Specifies the cursor drain meter config.

Customizable: Yes

Optional: No

Value: For example: <0x3 0x3>

window-fetch-meter-config

Description: Specifies the window fetch meter config.

Customizable: Yes

Optional: No

Value: For example: <0xf 0xf 0xf 0xf>

cursor-fetch-meter-config

Description: Specifies the cursor fetch meter config.

Customizable: Yes

Optional: No

Value: For example: <0x2 0x2>

start-fetch-delay-us:

Description: Specifies the delay before a START_FETCH command is sent to IsoHub.

Customizable: Yes

Optional: No

Value: For example: <0x1a8 0x1a8>

elv-start:

Description: Specifies the Elv start value.

Customizable: Yes

Optional: No

Value: For example: <0x4 0x4>

hub-clock-khz:

Description: Specifies the IsoHub clock frequency in KHz.

Customizable: Yes

Optional: No

Value: For example: <82300>

disp-clock-khz:

Description: Specifies the display clock frequency in KHz.

Customizable: Yes

Optional: No

Value: For example: <311862>

Serializer

Configuring the Serializer Driver:

To use the NVIDIA reference Maxim serializer driver, there are various ways to configure the driver by Device Tree. An example Device Tree fragment is shown below that configures the Maxim serializer chip in MST mode:

```
maxim_ser: max_gmsl_dp_ser@40 { compatible = "maxim,max_gmsl_dp_ser"; reg = <0x40>;
status = "okay";
max_gmsl_dp_ser-pwrdn = <&tegra_main_gpio TEGRA234_MAIN_GPIO(G, 3) GPIO_ACTIVE_HIGH>;
ser-errb = <&tegra_main_gpio TEGRA234_MAIN_GPIO(G, 7) 0>; dprx-link-rate = <0x1e>;
dprx-lane-count = <0x4>; enable-mst;
mst-payload-ids = <0x1 0x3 0x2 0x4>; gmsl-stream-ids = <0x0 0x1 0x2 0x3>; gmsl-link-
select = <0x0 0x0 0x1 0x1>; enable-dp-fec;
enable-dsc = <1 0>;
enable-gmsl-fec = <1 0>;
};
```

compatible:

Description: `compatible` contains the unique string to identify the Maxim Serializer DT node.

Customizable: No

Optional: No

Value: "maxim,max_gmsl_dp_ser"

reg:

Description: - I2C address of the Maxim display serializer.

Customizable: No

Optional: No

Value: <0x40>

max_gmsl_dp_ser-pwrdn:

Description: - GPIO pin number of the PWRDN pin. This pin is used to power up the Maxim display serializer chip.

Customizable: No

Optional: No

Value: TEGRA234_MAIN_GPIO(G, 3) GPIO_ACTIVE_HIGH

gmsl-link-select:

Description: - This property is an array of four unsigned 8-bit values that determines the GMSL output link to enable for each video pipe X, Y, Z, and U.

Customizable: Yes

Optional: No

Value: The possible values for each pipe are:

- > 0x0 (Link A)
- > 0x1 (Link B)
- > 0x2 (Link A + B)

For example: <0x0 0x0 0x1 0x1>

dprx-link-rate:

Description: Configures the DP link rate of the serializer chip.

Customizable: Yes

Optional: Yes

Value: The default value is 0x1E (HBR3). The possible values are:

- > 0xA (HBR)
- > 0x14 (HBR2)
- > 0x1E (HBR3)

dprx-lane-count:

Description: Configures the DP lane count of the serializer chip.

Customizable: Yes

Optional: Yes

Value: The default value is 0x4. The possible values are:

- > 0x1
- > 0x2
- > 0x4

ser-errb:

Description: - Specifies the GPIO pin number of the ERRB pin. This pin is used for error and fault reporting by the serializer chip.

Customizable: No

Optional: Yes

Value: TEGRA234_MAIN_GPIO(G, 7) 0

enable-mst:

Description: - Used to enable MST modes.

Customizable: No

Optional: Yes

Value: N/A

mst-payload-ids:

Description: Used to represent MST payload IDs of pipe X, Y, Z, U. This property is mandatory if enable-mst property is mentioned in dt.

Customizable: Yes

Optional: Yes

Value: It is an array of four unsigned 8-bit values.

For example: <0x1 0x3 0x2 0x4>

gmsl-stream-ids:

Description: Used to represent GMSL stream IDs of pipe X, Y, Z, U. This property is mandatory if enable-mst property is mentioned in dt.

Customizable: Yes

Optional: Yes

Value: It is an array of four unsigned 8-bit values.

For example: <0x0 0x1 0x2 0x3>

enable-dp-fec:

Description: Used to enable FEC on DP link if serializer supports it.

Customizable: No

Optional: Yes

Value: N/A

enable-dsc:

Description: Used to enable DSC.

Customizable: Yes

Optional: Yes

Value: It is an array of two 32-bit values, where each value indicates whether DSC is enabled or not. The first entry corresponds to video pipe X, and the second entry corresponds to video pipe Y. DSC is only supported on pipe X currently.

For example: <1 0>

enable-gmsl-fec:

Description: - Used to enable FEC on the GMSL link.

Customizable: Yes

Optional: Yes

Value: It is an array of two 32-bit values, where each value indicates whether FEC is enabled on the GMSL link. The first entry corresponds to GMSL Link A, and the second entry corresponds to GMSL Link B.

For example: <1 0>

Configuring video timings:

For both SST and MST mode, the mode timings that are used for each stream must be configured in Device Tree. Only one mode timing can be specified at a time for each video stream. The timings that are exposed in the EDIDs of the serializer and the panels connected to the downstream deserializer are completely ignored.

An example Device Tree fragment is shown below. In this example, a standard 1920x1080 at 60 Hz timing is specified for the first video stream, and a 1280x720 at 60 Hz timing is specified for the second video stream:

```
display-timings {
    display-connector-0 { dcb-index = <0>;

        stream-0 {
            timings-phandle = <&mode0>;
        };

        stream-1 {
            timings-phandle = <&mode1>;
        };
    };

    mode0: 1920-1080-60Hz {
        clock-frequency-khz = <148500>;
        hactive = <1920>;
        vactive = <1080>;
        hfront-porch = <88>;
        hback-porch = <148>;
        hsync-len = <44>;
        vfront-porch = <4>;
        vback-porch = <36>;
    };
}
```

```

vsync-len = <5>;
rrx1k = <60000>;
pps-data = [
11 00 00 89 30 80 04 38
07 80 04 38 03 c0 03 c0
02 00 03 58 00 20 73 3e
00 0d 00 0f 00 1d 00 0e
18 00 10 f0 03 0c 20 00
06 0b 0b 33 0e 1c 2a 38
46 54 62 69 70 77 79 7b
7d 7e 01 02 01 00 09 40
09 be 19 fc 19 fa 19 f8
1a 38 1a 78 22 b6 2a b6
2a f6 2a f4 43 34 63 74
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 ];
};

mode1: 1280-720-60Hz {
clock-frequency-khz = <74250>;
hactive = <1280>;
vactive = <720>;
hfront-porch = <110>;
hback-porch = <220>;
hsync-len = <40>;
vfront-porch = <5>;
vback-porch = <20>;
vsync-len = <5>;
rrx1k = <60000>;
};

```

display-timings:

Used to describe which timings are used for each stream.

display-connector-0:

The node specifies the timing information for the first display connector. If there are multiple display connectors present on the board that require fixed timings, then a new "display-connector" node must be created for each connector.

dcb-index:

Description: Specifies the logical index X of the DCB -> Display Devices -> Display Device X entry in the display DCB blob that this connector entry applies to.

Customizable: Yes

Optional: No

Value: Has to be a valid logical index. If there is only one display connector on the board, then "dcb-index" defaults to 0.

stream-0:

The nodes specify the phandle of the mode timing node that applies to the given video stream. Each "display-connector" can only have up to two (2) "stream" nodes. Note that it is fine to specify two (2) "stream" nodes even if the display serializer operates in SST mode because only the first "stream" node is consumed by the display driver. The extra node is ignored.

timings-phandle:

Description: Specifies the phandle of the mode timing node.

Customizable: Yes

Optional: No

Value: N/A

mode0:

Each "mode" node contains the actual mode timing parameters that will be used for a given video stream.

clock-frequency-khz:

Description: Specifies the pixel clock frequency in KHz.

Customizable: Yes

Optional: No

Value: For example: <148500>

hactive:

Description: Horizontal active pixels.

Customizable: Yes

Optional: No

Value: For example: <1920>

vactive:

Description: Vertical active pixels.

Customizable: Yes

Optional: No

Value: For example: <1080>

hfront-porch:

Description: Horizontal front porch.

Customizable: Yes

Optional: No

Value: For example: <88>

hback-porch:

Description: Horizontal back porch.

Customizable: Yes

Optional: No

Value: For example: <148>

hsync-len:

Description: Horizontal sync width.

Customizable: Yes

Optional: No

Value: For example: <44>

vfront-porch:

Description: Vertical front porch.

Customizable: Yes

Optional: No

Value: For example: <4>

hback-porch:

Description: Horizontal back porch.

Customizable: Yes

Optional: No

Value: Ex: <36>

vsync-len:

Description: Vertical sync width.

Customizable: Yes

Optional: No

Value: Ex: <5>

rrx1k:

Description: Refresh rates in units of 0.001Hz.

Customizable: Yes

Optional: No

Value: For example: <60000>

pps-data:

Description: All 128B of the DSC PPS.

Customizable: Yes

Optional: Yes

Value: This property should be specified if DSC will be enabled for the given timing. For example:

```
[  
11 00 00 89 30 80 04 38  
07 80 04 38 03 c0 03 c0  
02 00 03 58 00 20 73 3e  
00 0d 00 0f 00 1d 00 0e  
18 00 10 f0 03 0c 20 00  
06 0b 0b 33 0e 1c 2a 38  
46 54 62 69 70 77 79 7b  
7d 7e 01 02 01 00 09 40  
09 be 19 fc 19 fa 19 f8  
1a 38 1a 78 22 b6 2a b6  
2a f6 2a f4 43 34 63 74  
00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 ]
```

DCB Tool:

DCB tool is packaged in the following path:

`$(SDK_TOOLS_DIR)/dcb_tool/dcb_tool`

If you want to modify the DCB blob in DT, refer to the DCB documentation at:

`$(SDK_TOOLS_DIR)/dcb_tool/readme.txt`

14.2 NvGPU Device Tree

The NvGPU driver uses the following Device tree nodes and properties. Properties customizable are marked accordingly for each property used. Nodes used in NvGPU drivers for Linux are as follows.

Ga10b – ga10b node, contains configuration parameters needed for initializing the NvGPU driver.

```
tegra_ga10b: ga10b {
    compatible = "nvidia,ga10b";
    #cooling-cells = <2>;
    reg = <0x0 0x17000000 0x0 0x1000000
          0x0 0x18000000 0x0 0x1000000
          0x0 0x03b41000 0x0 0x00001000>;
    interrupts = <0 68 0x04
                 0 70 0x04
                 0 71 0x04
                 0 67 0x04>;
    dma-noncontig;
    interrupt-names = "stall0", "stall1", "stall2", "nonstall";
    nvidia,host1x = <&host1x>;
    access-vpr-phys;
    power-domains = <&bpmp TEGRA234_POWER_DOMAIN_GPU>;
    clocks = <&bpmp_clks TEGRA234_CLK_GPUSYS>,
              <&bpmp_clks TEGRA234_CLK_GPC0CLK>,
              <&bpmp_clks TEGRA234_CLK_GPC1CLK>;
    clock-names = "sysclk", "gpc0clk", "gpc1clk";
    resets = <&bpmp_resets TEGRA234_RESET_GPU>;
    dma-coherent;
    nvidia,bpmp = <&bpmp>;
    support-gpu-tools = <1>;
    status = "disabled";
};
```

Description: - `compatible` contains the unique string to identify the external NvGPU DT node.

Customizable: No

Optional: No

Value: " nvidia,ga10b "

Description: Physical base address and length of the controller's registers.

Customizable: No

Optional: No

Value: Must contain two entries:

- > first entry for bar0

- > second entry for bar1
- > third entry for FUSE region.

An optional third entry is used only in case of simulation.

Description: `interrupts` holds the IRQ number and IRQ type connect to ga10b.

Customizable: No

Optional: No

Value: Must contain an entry for each entry in interrupt-names.

- > 0 68 0x04
- > 0 70 0x04
- > 0 71 0x04
- > 0 67 0x04

Description: `interrupt-names` holds the type of interrupts supported by ga10b.

Customizable: No

Optional: No

Value: Must include the following entries:

- > stall0
- > stall1
- > stall2
- > nonstall

nvidia,host1x-

Description: host1x device, needed for syncpoint support.

Customizable: No

Optional: Yes

Value: host1x

access-vpr-phys

Description: GPU device can't access VPR via SMMU. It can only access VPR in physical. Bypass smmu if access-vpr-phys is specified in device's DT node. If device does not have DT entry or if it does not have access-vpr-phys property, nvmap can continue to map the sgt in iova space and hence can avoid fragmentation issues for devices which can access vpr through smmu

Customizable: No

Optional: Yes

Value: access-vpr-phys

power-domains-

Description: Generic Power Domains (genpd). Available in k4.14 onwards. bpmp manages power gating for gpu if this entry is present in the DT.

Customizable: No

Optional: No

Value:

```
<&bpmp TEGRA234_POWER_DOMAIN_GPU>
```

Clocks

Description: `clocks` holds the type of clocks ids supported by ga10b.

Customizable: No

Optional: No

Value:

```
<&bpmp_clks TEGRA234_CLK_GPUSYS>,
<&bpmp_clks TEGRA234_CLK_GPC0CLK>,
<&bpmp_clks TEGRA234_CLK_GPC1CLK
```

clock-names

Description: List of clock input name strings sorted in the same order as the clocks property. Consumer's drivers will use clock-names to match clock input names with clocks specifiers

Customizable: No

Optional: Yes

Value:

```
"sysclk", "gpc0clk", "gpc1clk
```

resets

Description: List of phandle and reset specifier pairs, one pair for each reset signal that affects the device, or that the device manages.

Customizable: No

Optional: No

Value:

```
bpmp_resets TEGRA234_RESET_GPU
```

dma-coherent

Description: Present if DMA operations are coherent.

Customizable: No

Optional: Yes

Value: N/A

support-gpu-tools

Description: Knob to control dbg/prof support.

Customizable: Yes

Optional: Yes

Value: Value 1 depicts that dbg/prof support is enabled. Absence or other value means support is disabled, which skips support for the following nodes:

1. ctxsw 2. dbg 3. prof 4. prof-dev 5. prof-ctx

Status

Description: `status` holds the NVGPU status.

Customizable: No

Optional: No

Value: "okay" or "disabled"

14.3 PCIe Controller Device Tree

This PCIe controller is based on the Synopsis Designware PCIe IP and thus inherits all the common properties defined in snps,dw-pcie.yaml and snps,dw-pcie-ep.yaml. Some of the controller instances are dual mode where in they can work either in root port mode or endpoint mode but one at a time.

Required Properties

- **power-domains**: A phandle to the node that controls power to the respective PCIe controller and a specifier name for the PCIe controller. Following are the specifiers for the different PCIe controllers:
 - TEGRA194_POWER_DOMAIN_PCIE8B: C0
 - TEGRA194_POWER_DOMAIN_PCIE1A: C1
 - TEGRA194_POWER_DOMAIN_PCIE1A: C2
 - TEGRA194_POWER_DOMAIN_PCIE1A: C3
 - TEGRA194_POWER_DOMAIN_PCIE4A: C4
 - TEGRA194_POWER_DOMAIN_PCIE8A: C5

These specifiers are defined in 20 "include/dt-bindings/power/tegra194-powergate.h" file.

- > **reg**: A list of physical base address and length pairs for each set of controller registers. Must contain an entry for each entry in the reg-names property.
- > **reg-names**: Must include the following entries:
 - "appl": Controller's application logic registers
 - "config": As per the definition in snps,dw-pcie.yaml.
 - "atu_dma": iATU and DMA registers. This is where the iATU (internal Address Translation Unit) registers of the PCIe core are made available for SW access.
 - "dbi": The aperture where root port's own configuration registers are available.
- > **interrupts**: A list of interrupt outputs of the controller. Must contain an entry for each entry in the interrupt-names property.
- > **interrupt-names**: Must include the following entries:
 - "intr": The Tegra interrupt that is asserted for controller interrupts.
- > **clocks**: Must contain an entry for each entry in clock-names.
- > **clock-names**: Must include the following entries:
 - core
- > **resets**: Must contain an entry for each entry in reset-names.
- > **reset-names**: Must include the following entries:
 - apb
 - core
- > **phys**: Must contain a phandle to P2U PHY for each entry in phy-names.
- > **phy-names**: Must include an entry for each active lane.
 - "p2u-N": where N ranges from 0 to one less than the total number of lanes
- > **nvidia,bpmp**: Must contain a pair of phandle to BPMP controller node followed by controller-id. The following are the controller IDs for each controller.
 - 0: C0
 - 1: C1
 - 2: C2
 - 3: C3
 - 4: C4
 - 5: C5
- > **vddio-pex-ctl-supply**: Regulator supply for PCIe side band signals.

RC Mode

- > **compatible**: Tegra19x must contain "nvidia,tegra194-pcie"
- > **device_type**: Must be "pci" for RC mode
- > **interrupt-names**: Must include the following entries: "msi": The Tegra interrupt that is asserted when an MSI is received
- > **bus-range**: Range of bus numbers associated with this controller
- > **#address-cells**: Address representation for root ports (must be 3):
 - cell 0 specifies the bus and device numbers of the root port:

[23:16]: bus number

[15:11]: device number

- cell 1 denotes the upper 32 address bits and should be 0
 - cell 2 contains the lower 32 address bits and is used to translate to the CPU address space.
- > **#size-cells**: Size representation for root ports (must be 2)
- > **ranges**: Describes the translation of addresses for root ports and standard PCI regions. The entries must be 7 cells each, where the first three cells correspond to the address as described for the #address-cells property above, the fourth and fifth cells are for the physical CPU address to translate to and the sixth and seventh cells are as described for the #size-cells property above.
- Entries setup the mapping for the standard I/O, memory and prefetchable PCI regions. The first cell determines the type of region that is setup:
 - 0x81000000: I/O memory region.
 - 0x82000000: non-prefetchable memory region.
 - 0xc2000000: prefetchable memory region.
- Refer to the standard PCI bus binding document for a more detailed explanation.
- > **#interrupt-cells**: Size representation for interrupts (must be 1).
- > **interrupt-map-mask** and **interrupt-map**: Standard PCI IRQ mapping properties. Refer to the standard PCI bus binding document for a more detailed explanation.

EP Mode

In Tegra194, Only controllers C0, C4, and C5 support EP mode.

- > **compatible**: Tegra19x must contain "nvidia,tegra194-pcie-ep"
- > **reg-names**: Must include the following entries: "addr_space": Used to map remote RC address space.
- > **reset-gpios**: Must contain a phandle to a GPIO controller followed by GPIO that is being used as PERST input signal. Refer to the pci.txt document.

Optional properties:

- > **pinctrl-names**: A list of pinctrl state names.

It is mandatory for C5 controller and optional for other controllers.

- "default": Configures PCIe I/O for proper operation.
 - > **pinctrl-0**: phandle for the 'default' state of pin configuration.
- It is mandatory for C5 controller and optional for other controllers.
- > **supports-clkreq**: Refer to Documentation/devicetree/bindings/pci/pci.txt
 - > **nvidia,update-fc-fixup**: This is a boolean property and needs to be present to improve performance when a platform is designed in such a way that it satisfies at least one of the following conditions thereby enabling root port to exchange optimum number of FC (Flow Control) credits with downstream devices.
 1. If C0/C4/C5 run at x1/x2 link widths (irrespective of speed and MPS)

2. If C0/C1/C2/C3/C4/C5 operate at their respective max link widths and:
 - a. speed is Gen-2 and MPS is 256B
 - b. speed is >= Gen-3 with any MPS
- > **nvidia,aspm-cmrt-us:** Common Mode Restore Time for proper operation of ASPM to be specified in microseconds.
 - > **nvidia,aspm-pwr-on-t-us:** Power On time for proper operation of ASPM to be specified in microseconds.
 - > **nvidia,aspm-l0s-entrance-latency-us:** ASPM L0s entrance latency to be specified in microseconds.
 - > **nvidia,pex-prsnt-gpios:** Must contain a phandle to a GPIO controller followed by GPIO that is being used to support hot plug and unplug via PRSNT# pin.
 - > **nvidia,enable-safety:** Set this property to enable safety features like HSI, etc.

RC Mode

- > **vpcie3v3-supply:** A phandle to the regulator node that supplies 3.3V to the slot if the platform has one such slot. (Ex:- x16 slot owned by C5 controller in p2972-0000 platform).
- > **vpcie12v-supply:** A phandle to the regulator node that supplies 12V to the slot if the platform has one such slot. (Ex:- x16 slot owned by C5 controller in p2972-0000 platform).

EP Mode

nvidia,refclk-select-gpios: Must contain a phandle to a GPIO controller followed by GPIO that is being used to enable REFCLK to controller from host.



Note: On Tegra194's P2972-0000 platform, only the C5 controller can be enabled to operate in the endpoint mode because of the way the platform is designed.

Examples

Tegra194 RC mode:

```
pcie@14180000 {
    compatible = "nvidia,tegra194-pcie";
    power-domains = <&bpmp TEGRA194_POWER_DOMAIN_PCIE8B>;
    reg = <0x00 0x14180000 0x0 0x00020000 /* appl registers (128K) */
          0x00 0x38000000 0x0 0x00040000 /* configuration space (256K) */
          0x00 0x38040000 0x0 0x00040000>; /* iATU_DMA reg space (256K) */
    reg-names = "appl", "config", "atu_dma";

    #address-cells = <3>;
    #size-cells = <2>;
    device_type = "pci";
    num-lanes = <8>;
    linux,pci-domain = <0>;

    pinctrl-names = "default";
    pinctrl-0 = <&pex_rst_c5_out_state>, <&clkreq_c5_bi_dir_state>;
```

```

clocks = <&bpmp TEGRA194_CLK_PEX0_CORE_0>;
clock-names = "core";

resets = <&bpmp TEGRA194_RESET_PEX0_CORE_0_APB>,
         <&bpmp TEGRA194_RESET_PEX0_CORE_0>;
reset-names = "apb", "core";

interrupts = <GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>, /* controller interrupt */
             <GIC_SPI 73 IRQ_TYPE_LEVEL_HIGH>; /* MSI interrupt */
interrupt-names = "intr", "msi";

#interrupt-cells = <1>;
interrupt-map-mask = <0 0 0 0>;
interrupt-map = <0 0 0 0 &gic GIC_SPI 72 IRQ_TYPE_LEVEL_HIGH>;

nvidia,bpmp = <&bpmp 0>;

supports-clkreq;
nvidia,aspm-cmrt-us = <60>;
nvidia,aspm-pwr-on-t-us = <20>;
nvidia,aspm-l0s-entrance-latency-us = <3>;

bus-range = <0x0 0xff>;
ranges = <0x81000000 0x0 0x38100000 0x0 0x38100000 0x0 0x00100000      /* downstream
I/O (1MB) */
          0x82000000 0x0 0x38200000 0x0 0x38200000 0x0 0x01E00000      /* non-prefetchable
memory (30MB) */
          0xc2000000 0x18 0x00000000 0x18 0x00000000 0x4 0x00000000>; /* prefetchable memory
(16GB) */

vddio-pex-ctl-supply = <&vdd_1v8ao>;
vpcie3v3-supply = <&vdd_3v3_pcie>;
vpcie12v-supply = <&vdd_12v_pcie>;

phys = <&p2u_hsio_2>, <&p2u_hsio_3>, <&p2u_hsio_4>,
       <&p2u_hsio_5>;
phy-names = "p2u-0", "p2u-1", "p2u-2", "p2u-3";
};

```

Tegra194 EP mode:

```

pcie_ep@141a000 {
    compatible = "nvidia,tegra194-pcie-ep", "snps,dw-pcie-ep";
    power-domains = <&bpmp TEGRA194_POWER_DOMAIN_PCIE8A>;
    reg = <0x00 0x141a0000 0x0 0x00020000  /* appl registers (128K) */
          0x00 0x3a040000 0x0 0x00040000  /* iATU_DMA reg space (256K) */
          0x00 0x3a080000 0x0 0x00040000  /* DBI reg space (256K) */
          0x1c 0x00000000 0x4 0x00000000>; /* Address Space (16G) */
    reg-names = "appl", "atu_dma", "dbi", "addr_space";

    num-lanes = <8>;
    num-ib-windows = <2>;
    num-ob-windows = <8>;
};

```

```

pinctrl-names = "default";
pinctrl-0 = <&clkreq_c5_bi_dir_state>;

clocks = <&bpmp TEGRA194_CLK_PEX1_CORE_5>;
clock-names = "core";

resets = <&bpmp TEGRA194_RESET_PEX1_CORE_5_APB>,
         <&bpmp TEGRA194_RESET_PEX1_CORE_5>;
reset-names = "apb", "core";

interrupts = <GIC_SPI 53 IRQ_TYPE_LEVEL_HIGH>; /* controller interrupt */
interrupt-names = "intr";

nvidia,bpmp = <&bpmp 5>;

nvidia,aspm-cmrt-us = <60>;
nvidia,aspm-pwr-on-t-us = <20>;
nvidia,aspm-l0s-entrance-latency-us = <3>;

vddio-pex-ctl-supply = <&vdd_1v8ao>;

reset-gpios = <&gpio TEGRA194_MAIN_GPIO(GG, 1) GPIO_ACTIVE_LOW>;

nvidia,refclk-select-gpios = <&gpio_aon TEGRA194_AON_GPIO(AA, 5)
                           GPIO_ACTIVE_HIGH>;

phys = <&p2u_nvhs_0>, <&p2u_nvhs_1>, <&p2u_nvhs_2>,
       <&p2u_nvhs_3>, <&p2u_nvhs_4>, <&p2u_nvhs_5>,
       <&p2u_nvhs_6>, <&p2u_nvhs_7>;

phy-names = "p2u-0", "p2u-1", "p2u-2", "p2u-3", "p2u-4",
            "p2u-5", "p2u-6", "p2u-7";
};


```

14.4 Camera Device Tree

The Device Tree stores platform-specific configurations for system resources used by Camera, which provides and regulates access to Camera FW resources, I2C, GPIO, powercontrol and synchronization signal generation functionalities among clients.

The properties are consumed by the Camera SW SIPL Camera Device Access Control (CDAC) resource manager on QNX, and its analogue on Linux, the Camera Linux KMD, composed of the cdi-mgr and cdi-tsc modules.

On DRIVE OS Linux, the platform configuration for Camera IP (such as RCE, NVCSI, VI, ISP) is part of the Host1x Device Tree node and documented in the NvMedia Device Tree Documentation. There is no equivalent Device Tree configuration on DRIVE OS QNX.

Resource Block Configuration

The `sipl_devblk_X` nodes contain the configuration for each resource block, which are software representations of deserializer groups in hardware. The `X` should denote the ID for the dedicated I2C bus of each deserializer block. GPIO pins for ordinary input and output, and interrupts are also configured in this node.

```

sipl_devblk_0 {
    status = "okay";
    compatible = "nvidia,cdi-mgr"; tegra {
        i2c-bus = <0>;
        csi-port = <0>;

        interrupt-parent = <&tegra_main_gpio>;
        interrupts = <TEGRA194_MAIN_GPIO(P, 5) 2>; /* GMSLA_STATUS_OC :
            falling edge sensitive */

        gpios {
            gpio@0 {
                index = <0>;
                nvgpio-line = <&nvgpio_errb_lock_err_a>; intr-edge-falling;
            };
            gpio@1 {
                index = <1>;
                nvgpio-line = <&nvgpio_cam_err_a>; intr-edge-rising;
            };
        };

        i2c_addr_pools {
            i2c-addrs-phys = <0x0 7>,
            <0x10 1>,
            <0x18 1>;
            i2c-addrs-virt = <0x43 12>,
            <0x57 8>,
            <0x65 12>;
        };

        deserializer {
            addr = <0x29>; /* 7 bit slave address */ reg_len = <16>; /* 16 bit register length */
            dat_len = <8>; /* 8 bit data length */
            des_i2c_port = <0>; /* I2C port number of the deserializer */
            /* des_tx_port = <0>; */ /* CSI Tx port number. Define des_tx_port only if the dedicated
               output Tx port is required */
            dphy_rate_x2 = <2500000>; /* Data rate in DPHY x2. Unit size is
               kbps */
            dphy_rate_x4 = <2500000>; /* Data rate in DPHY x4. Unit size is
               kbps */
            cphy_rate_x2 = <2000000>; /* Data rate in CPHY x2. Unit size is
               ksps */
            cphy_rate_x4 = <2000000>; /* Data rate in CPHY x4. Unit size is
               ksps */
        };
    };
}

```

```

gid = <3870>; /* Owning GID (Deserializer) */
links = <0 1 2 3>; /* Number and indices of valid links */ links_gid = <3870>; /* Owning
GID (Deserializer's links) */
};

pwr_ctrl {
power_port = <0>;
};
};

```

14.4.1 Common Properties

compatible:

Description: `compatible` contains the unique string to identify the cdi-mgr node.

Customizable: No

Optional: No

Value: "nvidia,cdi-mgr"

tegra node

i2c-bus:

Description: The system I2C bus number that the deserializer is physically connected to.

Customizable: Yes

Optional: No

Value: Typically one of {1, 3, 2, 7}

csi-port

Description: The NVCSI port that the deserializer is physically connected to.

Customizable: Yes

Optional: No

Value: NVCSI ports A-H, mapped by [0,7]

deserializer node

addr:

Description: The 7-bit I2C device address.

Customizable: Yes

Optional: No

Value: 0x29

reg_len

Description: The register length, should be set to 16 bits.

Customizable: Yes

Optional: No

Value: 16

dat_len:

Description: The data length, should be set to 8 bits.

Customizable: Yes

Optional: No

Value: 8

des_i2c_port:

Description: The I2C port on the deserializer that the deserializer is physically connected to. (Non-safety only.)

Customizable: Yes

Optional: Yes

Value: 0 or 1

des_tx_port

Description: The output CSI port on the deserializer. (Non-safety only.)

Customizable: Yes

Optional: Yes

Value: 0

default-reset-all

Description: Flag to set the Reset All register on the MAX96712 deserializer at Initialization. (Non-safety only.)

Customizable: Yes

Optional: Yes

Value: N/A

Data rates:

Properties with the name `'{dphy,cphy}_rate_{x2,x4}`.

Description: The data rate for a D-Phy or C-Phy link in x2 or x4 lane configuration in kb/s.
(Non-safety only.)

Customizable: Yes

Optional: Yes

Value: Valid integer data rate in kb/s in [0,UINT32_MAX]

pwr_ctrl node

deserializer-pwr-gpio:

Description: The deserializer power is controlled by GPIO

Customizable: No

Optional: No

Value: N/A

cam-pwr-max20087

Description: The external MAX20087 IC is used to control camera module link power, accessed over I2C.

Customizable: No

Optional: No

Value: N/A

tca9539 node

i2c-bus:

Description: The system I2C bus number that the TCA9539 IO expander is physically connected to.

Customizable: Yes

Optional: No

Value: Typically one of {1, 3, 2, 7}

addr:

Description: The 7-bit I2C device address.

Customizable: Yes

Optional: No

Value: 0x74

reg_len:

Description: The register length, should be set to 8 bits.

Customizable: No

Optional: No

Value: 8

dat_len:

Description: The data length, should be set to 8 bits.

Customizable: No

Optional: No

Value: 8

max20087 node**i2c-bus:**

Description: The system I2C bus number that the MAX20087 POC is physically connected to.

Customizable: Yes

Optional: No

Value: Typically one of {1, 3, 2, 7}

addr:

Description: The 7-bit I2C device address.

Customizable: Yes

Optional: No

Value: 0x28

reg_len:

Description: The register length, should be set to 8 bits.

Customizable: No

Optional: No

Value: 8

dat_len:

Description: The data length, should be set to 8 bits.

Customizable: No

Optional: No

Value: 8

links:

Description:Vector of camera module to power link control mappings.

Customizable: Yes

Optional:No

Value:<0, 1, 2, 3>

14.4.2 Linux Properties

Property	Description	Customizable	Optional	Value
pwdn-gpios	Should contain the list for GPIOs to control power in order of aggregator and camera instances, tuples contain the GPIO interrupt number and triggering mode flag.	Yes	Yes	N/A
pwr-items	Map power items to the GPIO item specified in the pwdn-gpios list, starting from 0. For example, pwr-item 0 is for the deserializer, while pwr-item 1, 2, 3, 4 are for links 0, 1, 2, 3.	Yes	Yes	N/A
default-power-on	If this flag is present, while probing the device, pwdn-gpios are powered on.	Yes	Yes	N/A

Property	Description	Customizable	Optional	Value
runtime-pwrctrl-off	If this property is present, power control GPIOs retain the same status and are never changed.	Yes	Yes	N/A
interrupt-parent	Should contain interrupt parent for GPIO interrupt (phandle).	Yes	Yes	<&tegra_main_gpio>
interrupts	Should contain the GPIO interrupt number and triggering mode flag.	Yes	Yes	<TEGRA194_MAIN_GPIO(P, 5) 2>

pwr_ctrl node

Property	Description	Customizable	Optional	Value
power_port	The GPIO expander port that controls deserializer power. Required if the tca9539 node is present.	No	No	0

14.4.3 Fsync Signal Generation

Unset:

```
tsc_sig_gen@c6a0000 {
    compatible = "nvidia,tegra234-cdi-tsc";
    ranges = <0x0 0x0 0xc6a0000 0x10000>;
    reg = <0x0 0xc6a0000 0x0 0x18>;
    #address-cells = <1>;
    #size-cells = <1>;
    status = "okay";

    gen0: generator@380 {
        reg = <0x380 0x80>;
        freq_hz = <30>;
        duty_cycle = <25>;
        offset_ms = <0>;
        status = "okay";
    };
};
```

```

gen1: generator@400 {
    reg = <0x400 0x80>;
    freq_hz = <30>;
    duty_cycle = <25>;
    offset_ms = <0>;
    status = "okay";
};

gen2: generator@480 {
    reg = <0x480 0x80>;
    freq_hz = <30>;
    duty_cycle = <25>;
    offset_ms = <10>;
    status = "okay";
};

gen3: generator@500 {
    reg = <0x500 0x80>;
    freq_hz = <60>;
    duty_cycle = <25>;
    offset_ms = <20>;
    status = "okay";
};
};

fsync-groups {
    status = "disabled";
    fsync-group@0 {
        id = <0>;
        status = "okay";
        generators = <&gen0>, <&gen2>, <&gen3>;
    };
    fsync-group@1 {
        id = <1>;
        status = "okay";
        generators = <&gen1>;
    };
};

```

Common Properties

Property	Description	Customizable	Optional	Value
compatible	compatible contains the unique string to identify the fsync node.	No	No	nvidia,tegra234- cdi-tsc

generator@Y nodes

The `generator@Y` nodes contain the configuration for individual generated signals. The `Y` should denote the base address for this line.

Property	Description	Customizable	Optional	Value
freq_hz	Frequency of the signal, in hertz.	Yes	No	Hertz as an unsigned integer in the range (0,120]
duty_cycle	Percentage duty cycle of the signal.	Yes	No	Unsigned in the range (0, 100)
offset_ms	Offset to shift the signal, in milliseconds.	Yes	No	Milliseconds as an unsigned integer in the range [0, 1000]

Fsync-groups

1. Fsync-groups allow users to logically group individual generators based on intended usage. All generators in a group are always started together. Any offsets are relative to only generators in that group. Generators in other groups may be independently started.
2. Consider the ffsync-groups in the device tree fragment above:
 - a. All generators in group #0 will be started at the same time. Generators #2 and #3 will lag 10 and 20ms relative to Generator #0, respectively.
 - b. The generator in group #1 can be independently started. It need not necessarily be synchronized with generators in group #0.
3. It is mandatory that each generator is only part of one group.

Includes subnodes for each ffsync-group:

Property	Description	Customizable	Optional	Value
Status	If ffsync-groups are enabled or disabled.	Yes	No	If okay, ffsync-groups is used and active groups can be programmed to start generators within the group at a given time. Otherwise, all active generators start with the default start time on startup.

fsync-group@X

The `fsync-group@X` nodes contains the configuration for individual fsync-group. The `X` should denote the index of the group.

Table 7.

Property	Description	Customizable	Optional	Value
id	ID of group.	Yes	No	Unique index as an unsigned integer, starting from index 0.
status	Specifies whether a group is enabled or disabled.	Yes	No	Okay if group is enabled, otherwise disabled.
generators	List of phandles of the generators.	Yes	No	List of phandles of all the generators within the group. All generators within the group are in sync with each other.

14.5 NvHost Device Tree

NvHost element has a resource manager deployed in Guest VM and relies on the Device Tree to fetch the platform configuration.

This document lists down the Device Tree nodes and properties used by the NvHost element. Properties customizable are marked accordingly for each property used.

14.5.1 Host1x Device Node

The Device Tree Node contains configuration parameters required to initialize and provide the functionality of the NvHost element in the Guest VM.

```
host1x: host1x@13e00000 {
    compatible = "nvidia,tegra234-host1x", "simple-bus"; reg = <0x0 0x13e40000 0x0
    0x00010000>,
    <0x0 0x13e10000 0x0 0x00010000>,
    <0x0 0x13ef0000 0x0 0x00040000>,
    <0x0 0x60000000 0x0 0x04000000>,
    <0x0 0x13e00000 0x0 0x00010000>;
    reg-names = "guest", "hypervisor", "actmon", "sem-syncpt-shim", "common";
    interrupts = <0 448 0x04>,
```

```

<0 449 0x04>,
<0 450 0x04>,
<0 451 0x04>,
<0 452 0x04>,
<0 453 0x04>,
<0 454 0x04>,
<0 455 0x04>,
<0 263 0x04>;
nvidia,vmid = <1>;

nvidia,host1x-ctx-streamid = <&smmu_niso0 TEGRA_SID_NIS00_HOST1X_CTX0
&smmu_niso1 TEGRA_SID_NIS01_HOST1X_CTX0>;

nvidia,ch-base = <0>;
nvidia,nb-channels = <47>;

nvidia,nb-hw-pts = <1024>;
nvidia,pts-base = <0>;
nvidia,nb-pts = <576>;

nvidia,gpu-syncpt-pool = <0 128>;
nvidia,vi-syncpt-pool = <288 144>;
vmserver-owns-engines = <1>; ivc-queue0 = <&tegra_hv 42>;

iommu = <&smmu_niso1 TEGRA_SID_NIS01_HC>; status = "okay";

host1x_ctx0: niso0_ctx0 {
compatible = "nvidia,tegra186-iommu-context";
iommu = <&smmu_niso0 TEGRA_SID_NIS00_HOST1X_CTX0>; status = "okay";
};

host1x_ctx0n1: niso1_ctx0 {
compatible = "nvidia,tegra186-iommu-context";
iommu = <&smmu_niso1 TEGRA_SID_NIS01_HOST1X_CTX0>; status = "okay";
};

vic@15340000 {
compatible = "nvidia,tegra234-vic";
iommu = <&smmu_niso1 TEGRA_SID_NIS01_VIC>; status = "okay";
};

nvjpg@15380000 {
compatible = "nvidia,tegra234-nvjpg";
iommu = <&smmu_niso1 TEGRA_SID_NIS01_NVJPG>; status = "okay";
};

nvjpg1@15540000 {
compatible = "nvidia,tegra234-nvjpg";
iommu = <&smmu_niso0 TEGRA_SID_NIS00_NVJPG1>; status = "okay";
};

tsec@15500000 {
compatible = "nvidia,tegra234-tsec";
iommu = <&smmu_niso1 TEGRA_SID_NIS01_TSEC>; status = "okay";
};

```

```

};

nvdec@15480000 {
    compatible = "nvidia,tegra234-nvdec";
    iommus = <&smmu_niso1 TEGRA_SID_NIS01_NVDEC>; status = "okay";
};
nvenc@154c0000 {
    compatible = "nvidia,tegra234-nvenc";
    iommus = <&smmu_niso0 TEGRA_SID_NIS00_NVENC>; status = "okay";
};

ofa@15a50000 {
    compatible = "nvidia,tegra234-ofa";
    iommus = <&smmu_niso0 TEGRA_SID_NIS00_OFA>; status = "okay";
};
};

```

Properties

Following are properties used across different platform configurations.

Compatibility String "compatible"

Description	Strings specifying the compatibility of the Host1x Device Node
Customizable	No
Type	String
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	"nvidia,tegra234-host1x-hv"

Virtual Machine Host1x ID "NVIDIA,vmid"

Description	ID corresponding to the Host1x guest page allocated to the Guest VM
Customizable	No
Type	Unsigned integer
Unit	NA

Description	ID corresponding to the Host1x guest page allocated to the Guest VM
Optional	No
Resolution	1
MaxValue	8
MinValue	1
Recommended	Based on the platform configuration

Register Names "reg-names"

Description	List of Host1x MMIO register page names, which can be accessed by NvHost from the Guest VM
Customizable	No
Type	List<String>
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	<p><AV+Q configuration>:</p> <p>"guest", "sem-syncpt-shim";</p> <p><AV+L configuration>:</p> <p>"guest", "hypervisor", "actmon", "sem-syncpt-shim", "common";</p>

Register Apertures "reg"

Description	List of Host1x MMIO register page apertures allowed to be accessed by NvHost from the Guest VM. Type of page for each aperture is identified by the “reg-names” entry of the same index
Customizable	No
Type	List<Aperture start, Aperture Size>
Unit	NA
Optional	No
Resolution	NA
.MaxValue	NA
.MinValue	NA
Recommended	Values fetched from the Host1x HW IAS

Interrupts "interrupts"

Description	List of Host1x syncpoint interrupt lines assigned to the Guest VM
Customizable	No
Type	List<IRQ number>
Unit	NA
Optional	No
Resolution	NA
.MaxValue	NA
.MinValue	NA
Recommended	Values fetched from the Host1x HW IAS based on platform configuration.

Host1x Shared StreamID "nvidia,host1x-ctx-streamid"

Description	FallbackStreamId to access the data buffers in case context pools are not defined by the platform configuration
Customizable	No
Type	<SMMUinstance, StreamId>
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	Based on the platform configuration

Host1x Channel Pool Base "nvidia,ch-base"

Description	Base of the Host1x channel pool which can be accessed from the Guest VM via the NvHost
Customizable	Yes
Type	Unsigned integer
Unit	NA
Optional	No
Resolution	NA
MaxValue	62
MinValue	0
Recommended	Based on the platform configuration

Host1x Channel Pool Count "nvidia,nb-channels"

Description	Size of the Host1x channel pool which can be accessed from the Guest VM via the NvHost
Customizable	Yes
Type	Unsigned integer

Description	Size of the Host1x channel pool which can be accessed from the Guest VM via the NvHost
Unit	NA
Optional	No
Resolution	NA
.MaxValue	63
.MinValue	0
Recommended	Based on the platform configuration

Host1x Syncpoint Total Count "nvidia,nb-hw-pts"

Description	Total number of Host1x HW syncpoints available in the SoC
Customizable	No
Type	Unsigned integer
Unit	NA
Optional	No
Resolution	NA
.MaxValue	NA
.MinValue	NA
Recommended	1024

Host1x Syncpoint Pool Base "nvidia,pts-base"

Description	Base of the Host1x syncpoints pool, which can be incremented from the Guest VM
Customizable	Yes
Type	Unsigned integer
Unit	NA
Optional	No
Resolution	NA

Description	Base of the Host1x syncpoints pool, which can be incremented from the Guest VM
MaxValue	1023
MinValue	0
Recommended	Based on platform configuration

Host1x Syncpoint Pool Count "nvidia,nb-pts"

Description	Size of the Host1x syncpoints pool, which can be incremented from the Guest VM
Customizable	Yes
Type	Unsignedinteger
Unit	NA
Optional	No
Resolution	NA
MaxValue	1024
MinValue	0
Recommended	Based on platform configuration

Host1x Syncpoint Pool for GPU "nvidia,gpu-syncpt-pool"

Description	Host1x syncpoint pool, which is write accessible from the GPU and allocated exclusively for GPU use cases. The pool is outside of the generic pool <i>Note: Only applicable for the AV+Q Safety configuration</i>
Customizable	Yes
Type	<Poolbase, Pool size>
Unit	NA
Optional	No
Resolution	NA

Description	Host1x syncpoint pool, which is write accessible from the GPU and allocated exclusively for GPU use cases. The pool is outside of the generic pool Note: <i>Only applicable for the AV+Q Safety configuration</i>
MaxValue	NA
MinValue	NA
Recommended	Based on platform configuration

Host1x Syncpoint Pool for VI "nviida,vi-syncpt-pool"

Description	Host1x syncpoints pool ,which is exclusively allocated for VI use cases. The pool is within the generic pool Note: <i>Only applicable to the AV+Q Safety configuration</i>
Customizable	Yes
Type	<Poolbase, Pool size>
Unit	NA
Optional	No
Resolution	NA
.MaxValue	NA
.MinValue	NA
Recommended	Based on platform configuration

Engine Ownership Flag "vmserver-owns-engines"

Description	Flag specifying whether the engine ownership belongs to NvHost Server Note: <i>Only applicable to AV+L configuration</i>
Customizable	No
Type	Unsigned integer

Description	Flag specifying whether the engine ownership belongs to NvHost Server <i>Note: Only applicable to AV+L configuration</i>
Unit	NA
Optional	No
Resolution	NA
MaxValue	1
MinValue	0
Recommended	1

IVC Queue Identifier "ivc-queue0"

Description	Unique identifier of the IVC Queue created to communicate with the NvHost Server
Customizable	No
Type	UnsignedInteger
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	Based on the platform configuration

Host1x StreamId Info "iommu"

Description	Set of stream Id info, which is assigned for Host1x to perform DMA. StreamId used from the Guest VM is derived based on the VMID
Customizable	No
Type	List<SMMU instance, Stream Id>
Unit	NA

Description	Set of stream Id info, which is assigned for Host1x to perform DMA. StreamId used from the Guest VM is derived based on the VMID
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	Based on platform configuration

Host1x Device Status Flag "status"

Description	Flag specifying whether the Host1x device is enabled
Customizable	Yes
Type	String
Unit	NA
Optional	No
Resolution	NA
MaxValue	“disabled”
MinValue	“okay”
Recommended	“okay”

Host1x Context Node Entry "host1x_ctx"

Description	Device Tree Node specifying the properties of the Host1x contexts, which are allocated to the VM
Customizable	Yes
Type	Host1xContext Node
Unit	NA
Optional	No
Resolution	NA

Description		Device Tree Node specifying the properties of the Host1x contexts, which are allocated to the VM
MaxValue		NA
MinValue		NA
Recommended		Based on platform configuration

VIC Engine Node "vic"

Description		Device Tree Node specifying the properties of the VIC engine needed for NvHost Driver
Customizable		Yes
Type		Host1xEngine Node
Unit		NA
Optional		No
Resolution		NA
MaxValue		NA
MinValue		NA
Recommended		<i>compatible = "nvidia,tegra234-vic";</i> <i>iommus= <&smmu_niso1 TEGRA_SID_NISO1_VIC>;</i> <i>status = <Based on platform configuration>;</i>

NVJPG Engine Node "nvjpg"

Description		Device Tree Node specifying the properties of the first instance of NVJPG engine needed for NvHost Driver
Customizable		Yes
Type		Host1xEngine Node
Unit		NA
Optional		No
Resolution		NA

Description	Device Tree Node specifying the properties of the first instance of NVJPG engine needed for NvHost Driver
MaxValue	NA
MinValue	NA
Recommended	<code>compatible = "nvidia,tegra234-nvjpg"; iommu= <&smmu_niso1 TEGRA_SID_NISO1_NVJPG>; status = <Based on platform configuration>;</code>

NVJPG1 Engine Node "nvjpg1"

Description	Device Tree Node specifying the properties of the second instance of NVJPG engine needed for NvHost Driver
Customizable	Yes
Type	Host1xEngine Node
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	<code>compatible = "nvidia,tegra234-nvjpg"; iommu= <&smmu_niso0 TEGRA_SID_NISO0_NVJPG1>; status = <Based on platform configuration>;</code>

NVDEC Engine Node "nvdec"

Description	Device Tree Node specifying the properties of the NVDEC engine needed for NvHost Driver.
Customizable	Yes
Type	Host1xEngine Node
Unit	NA
Optional	No

Description	Device Tree Node specifying the properties of the NVDEC engine needed for NvHost Driver.
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	<pre>compatible = "nvidia,tegra234-nvdec"; iommu= <&smmu_niso1 TEGRA_SID_NISO1_NVDEC>; status = <Based on platform configuration>;</pre>

NVENC Engine Node "nvenc"

Description	Device Tree Node specifying the properties of the NVENC engine needed for NvHost Driver.
Customizable	Yes
Type	Host1xEngine Node
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	<pre>compatible = "nvidia,tegra234-nvenc"; iommu= <&smmu_niso0 TEGRA_SID_NISO0_NVENC>; status = <Based on platform configuration>;</pre>

OFA Engine Node "ofa"

Description	DeviceTree Node specifying the properties of the OFA engine needed for NvHost Driver.
Customizable	Yes
Type	Host1xEngine Node
Unit	NA
Optional	No

Description	DeviceTree Node specifying the properties of the OFA engine needed for NvHost Driver.
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	<code>compatible = "nvidia,tegra234-ofa"; iommu= <&smmu_niso0 TEGRA_SID_NISO0_OFA>; status = <Based on platform configuration>;</code>

TSEC Engine Node "tsec"

Description	Device Tree Node specifying the properties of the TSEC engine needed for NvHost Driver. <i>Note: Only applicable for AV+L configuration</i>
Customizable	Yes
Type	Host1xEngine Node
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	<code>compatible = "nvidia,tegra234-tsec"; iommu= <&smmu_niso1 TEGRA_SID_NISO1_TSEC>; status = <Based on platform configuration>;</code>

14.5.2 Host1x Context Node

Device Tree Node node contains properties to configure Host1x contexts in a Guest VM.

```
host1x_ctx {
    compatible = "nvidia,tegra186-iommu-context";
    iommus = <&smmu_niso0 TEGRA_SID_NISO0_HOST1X_CTX0>; status = "okay";
};
```

Properties

This section lists the properties that are used across different platform configurations.

Compatibility String "compatible"

Description	String specifying the compatibility of the Host1x Context Entry
Customizable	No
Type	String
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	"nvidia,tegra186-iommu-context"

Context StreamId "iommu"

Description	StreamId associated with the Host1x context.
Customizable	No
Type	<SMMUinstance, Stream Id>
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	Based on platform configuration

Host1x Context Status Flag "status"

Description	Flag specifying whether a specific Host1x context is enabled.
Customizable	Yes
Type	String

Description	Flag specifying whether a specific Host1x context is enabled.
Unit	NA
Optional	No
Resolution	NA
MaxValue	“disabled”
MinValue	“okay”
Recommended	“okay”

14.5.3 Host1x Engine Node

Device Tree Node node, which contains properties to configure a Host1x client engine allocated to the Guest VM.

```
<engine-name> {
    compatible = "nvidia,tegra234-tsec";
    iommus = <&smmu_niso1 TEGRA_SID_NIS01_TSEC>; status = "okay";
};
```

Properties

This section lists the properties used across different platform configurations.

Compatibility String "compatible"

Description	String specifying the compatibility of the Host1x Engine Entry
Customizable	No
Type	String
Unit	N/A
Optional	No
Resolution	N/A
MaxValue	N/A
MinValue	N/A
Recommended	“nvidia,tegra234-<engine_name>”

Engine StreamId "iommus"

Description	StreamId associated with the Host1x engine per HW specification.
Customizable	No
Type	<SMMUinstance, Stream Id>
Unit	N/A
Optional	No
Resolution	N/A
MaxValue	N/A
MinValue	N/A
Recommended	Based on the platform configuration

Host1x Engine Status Flag "status"

Description	Flag specifying whether a specific Host1x engine is enabled.
Customizable	Yes
Type	String
Unit	N/A
Optional	No
Resolution	N/A
MaxValue	“disabled”
MinValue	“okay”
Recommended	“okay”

14.5.4 NvHost FSW Attributes Node

Device Tree Node node contains properties to configure the NvHost Fast Syncpoint Wait functionality.

```
nvhost_fsw_attributes {
    compatible = "nvidia,nvhost_fsw_attributes"; nvhost_type_id = <1>;
    status = "okay";
};
```

Properties

This section lists the properties that are used across different platform configurations.

Compatibility String "compatible"

Description	String specifying the compatibility of the NvHost FSW Attributes Entry
Customizable	No
Type	String
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	"nvidia,nvhost_fsw_attributes"

NvHost FSW Type ID "nvhost_type_id"

Description	QNX process type Id statically assigned to the NvHost resource manager. <i>Note:Applicable only to AV+Q configuration</i>
Customizable	No
Type	Unsigned integer
Unit	NA
Optional	No
Resolution	NA
MaxValue	NA
MinValue	NA
Recommended	1

NvHost FSW Attributes Status Flag "status"

Description	Flag specifying whether the NvHost FSW Attributes node is enabled.
Customizable	Yes
Type	String
Unit	NA
Optional	No
Resolution	NA
MaxValue	“disabled”
MinValue	“okay”
Recommended	“okay”

14.6 DLA Device Tree

The nvdla0 and nvdla1 nodes contain the configuration parameters needed for initializing the DLA driver.

Sample Device Tree

This section provides the sample entries for the nvdla0 and nvdla1 nodes.

```

nvdla0: nvdla0@15880000 {
    compatible = "nvidia,tegra234-nvdla";
    power-domains = <&bpmp TEGRA234_POWER_DOMAIN_DLAA>;
    reg = <0x0 0x15880000 0x0 0x00040000>;
    interrupts = <0 236 0x04>;

    resets = <&bpmp_resets TEGRA234_RESET_DLA0>;
    clocks = <&bpmp_clks TEGRA234_CLK_DLA0_CORE>,
              <&bpmp_clks TEGRA234_CLK_DLA0_FALCON>;
    clock-names = "nvdla0", "nvdla0_flcn";

    iommus = <&smmu_niso1 TEGRA_SID_NIS01_NVDLA0>;
    dma-coherent;
    status = "okay";
};

nvdla1: nvdla1@158c0000 {
    compatible = "nvidia,tegra234-nvdla";
    power-domains = <&bpmp TEGRA234_POWER_DOMAIN_DLAB>;
    reg = <0x0 0x158c0000 0x0 0x00040000>;
    interrupts = <0 237 0x04>;
};

```

```

    resets = <&bpmp_resets TEGRA234_RESET_DLA1>;
    clocks = <&bpmp_clks TEGRA234_CLK_DLA1_CORE>,
              <&bpmp_clks TEGRA234_CLK_DLA1_FALCON>;
    clock-names = "nvdla1", "nvdla1_flcn";

    iommus = <&smmu_niso0 TEGRA_SID_NIS00_NVDLA1>;
    dma-coherent;
    status = "okay";
};

}

```

DLA Device Tree Properties

Property	Description	Customizable	Optional	Value
compatible	A unique string to identify the DLA DT node	No	No	nvidia,tegra234-nvdla
power-domains	A phandle and PM domain specifier as defined by bindings of the power controller specified by phandle	No	No	TEGRA234_POWER_DOMAIN_DLA[A B]
reg	Physical base address and length of the controller's registers	No	No	Must contain two entries: one for the physical address and other for the length.
interrupts	Describes the IRQ number and IRQ type	No	No	Must contain an entry for each interrupt.
resets	List of phandle and reset specifier pairs, one pair for each reset signal that affects the device, or that the device manages	No	No	TEGRA234_RESET_DLA[0 1]
clocks	the type of clocks ids that are supported	No	No	TEGRA234_CLK_DLA[0 1]_CORE and TEGRA234_CLK_DLA[0 1]_FALCON

Property	Description	Customizable	Optional	Value
clock-names	List of clock input name strings sorted in the same order as the clocks property. Consumer's drivers will use clock-names to match clock input names with clocks specifiers	No	Yes	nvdla[0 1] and nvdla[0 1]_flcn
iommus	A list of phandle and IOMMU specifier pairs that describe the IOMMU master interfaces of the device. One entry in the list describes one master interface of the device	No	No	TEGRA_SID_NISO1_NVDA0, TEGRA_SID_NISO0_NVDA1
dma-coherent	Present if DMA operations are coherent	No	No	N/A
status	status of NVDLA nodes	No	Yes	okay or disabled

Chapter 16. NVIDIA DRIVE OS 6.0

Third-Party Software Licenses

Introduction

NVIDIA DRIVE® OS 6.0 software contains Third-Party components that are provided for internal, non-commercial use only. Developers are exclusively responsible for obtaining any and all authorizations and licenses required for development and for distribution and/or incorporation of the applicable Third-Party components.

**Note:**

The following information may be revised prior to production.

Follow the instructions below to locate the Open Source Software (OSS) Source utilized by DRIVE OS.

File name:

`nv-driveos-linux-oss-src-6.0.X.0-<GCID>_6.0.X.0-<GCID>_amd64.deb`

`nv-driveos-foundation-oss-src-6.0.X.0-<GCID>_6.0.X.0-<GCID>_amd64.deb`

Location:

- > For Docker Containers:

`/drive/drive-linux_src <SDK/PDK>`

`/drive/drive-foundation_src <PDK>`



Note: In NVIDIA Docker installations, `NV_WORKSPACE=/drive` by default.

- > For SDK Manager:

`<install_path>/DRIVE_OS_6.0.x_SDK_Linux_DRIVE_AGX_ORIN_DEVKITS/DRIVEOS/drive-linux_src`

`<install_path>/DRIVE_OS_6.0.x_SDK_Linux_DRIVE_AGX_ORIN_DEVKITS/DRIVEOS/drive-foundation_src`

Where `install_path` is the path chosen in Step 2 for "Target HW image folder".

- > For Debian Packages:

\$NV_WORKSPACE/drive-linux_src <SDK/PDK>

\$NV_WORKSPACE/drive-foundation_src <PDK>

NVIDIA Third-Party Free Open Source Software (FOSS) Licenses

This section lists the Free Open Source Software used in the development of the platform.

[NVIDIA DRIVE OS 6.0 Free Open Source Software \(FOSS\) Licenses report for Linux Standard release](#)

[DRIVE OS Free Open Source Software \(FOSS\) Supplemental License Catalog](#)

NVIDIA Third-Party Proprietary Software Licenses

This section lists the Third-Party components used in the development of the platform.

Note: The following Third-Party Companies require further licensing for development and/or production.

Company	Component	Type
Vector Informatik GmbH	AutoSAR Classic	Firmware
Infineon Technologies	AutoSAR MCAL	MCAL / Driver
ARM Compiler	Compiler	Tools

Note: The following Third-Party Company requires no further licensing.

Company	Component	Type
Marvell Technology Group	Automotive Ethernet Switch 88Q5050	Driver

Marvel Technology Group License Text

Marvell International Ltd.

```
/*
 *      LICENSE:
 *      (C)Copyright 2010-2011 Marvell.
 *
 *      All Rights Reserved
 *
 *      THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF MARVELL
 *      The copyright notice above does not evidence any
 *      actual or intended publication of such source code.
 *
 *      This Module contains Proprietary Information of Marvell
```

```
*      and should be treated as Confidential.  
*  
*      The information in this file is provided for the exclusive use of  
*      the licensees of Marvell. Such users have the right to use, modify,  
*      and incorporate this code into products for purposes authorized by  
*      the license agreement provided they include this notice and the  
*      associated copyright notice with any such product.  
*  
*      The information in this file is provided "AS IS" without warranty.  
*  
*      Contents: functions as called from Linux kernel (entry points)  
*  
* Path  : OAK-Linux::OAK Switch Board - Linux Driver::Class Model::oak  
* Author: afischer  
* Date   :2020-05-06  - 11:03  
* */
```

Chapter 17. Legal Information

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OR CONDITION OF TITLE, MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT, ARE HEREBY EXCLUDED TO THE MAXIMUM EXTENT PERMITTED BY LAW.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA, the NVIDIA logo, CUDA, Jetson, NVIDIA DRIVE, Tegra, and TensorRT are trademarks or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

ARM, AMBA, and ARM Powered are registered trademarks of ARM Limited. Cortex, MPCore and Mali are trademarks of ARM Limited. "ARM" is used to represent ARM Holdings plc; its operating company ARM Limited; and the regional subsidiaries ARM Inc.; ARM KK; ARM Korea Limited.; ARM Taiwan Limited; ARM France SAS; ARM Consulting (Shanghai) Co. Ltd.; ARM Germany GmbH; ARM Embedded Technologies Pvt. Ltd.; ARM Norway, AS and ARM Sweden AB.

BLACKBERRY, EMBLEM Design, QNX, AVIAGE, MOMENTICS, NEUTRINO and QNX CAR are the trademarks or registered trademarks of BlackBerry Limited, used under license, and the exclusive rights to such trademarks are expressly reserved.

Copyright

© 2023 by NVIDIA Corporation and Affiliates. All rights reserved.