NVIDIA DRIVE OS 6.0 SDK Migration Guide

Version: 6.0.9.2

Document History

Version	Date	Description of Change
01	August 1, 2021	Initial release
02	August 16, 2021	> Added Installation and Tools Changes chapter > Added NvMedia API Changes section > Added Drive Update Changes section > Added the following sections to the TensorRT content: • Symbols • Plugins and Plugin Registry • Library Names
03	August 24, 2021	In the TensorRT API Changes section, added information about a build flag to enable safety restrictions and a method check for IBuilder.
04	September 30, 2021	> Incorporated major content and organizational changes to NvMedia API tables > Incorporated updates to the TensorRT API Changes section > Updated the last four rows in the SDK File Locations in 5.2 and 6.0 table
05	December 16, 2021	> Updated content in the NvStreams API Changes section > Updated the Tensor RT API Changes section to rename the section and remove content now found in the release notes. > Added a table: TensorRT Changes for 6.0.1 and 6.0.2
06	January 21, 2022	> Added the following sections: • NvMedia Image Processing Pipeline (IPP) Changes Introduced in DRIVE OS 5.1.15.0 • NvMedia 2D: Migrating from 5.2 to 6.0 • NvMedia Lens Distortion Correction (LDC): Migrating from 5.2 to 6.0 > Updated TensorRT information > Implemented editorial updates
07	February 11, 2022	> Added a timeline summary of NvMedia API changes
08	March 10, 2022	> Added the NvScilpc API Changes section

Version	Date	Description of Change
09	April 12, 2022	> Added the Linux targetfs directory to targetfs image section > Added information about a TensorRT API
10	July, 2022	> Added PKCS#11 API Changes section > Added Board Support Packages (BSP) API Changes > Added a Utilities section > Updated the NvScilpc API Changes section for 6.0.4 > Updated the NvMedia Core Deprecation section > Added NvMedia Tensor API Changes section
11	September, 2022	> Added information about nvGPIO to the BSP Driver table > Updated PKCS#11 information for 6.0.5 > Updated code snippets in the Camera/SIPL API Changes table
12	January, 2023	> Added a section: NITO File Path Changes in 6.0 > Added a topic on Accessing and Choosing a Token > Updated NvSciplc Deprecated and Modified APIs section
13	March, 2023	> Added NvMediaEncodeQuality Deprecation section
14	April, 2023	> Updated TensorRT bulleted list in DRIVE OS 5.2 to 6.0 Packaging Changes section > Updated Camera/SIPL API Changes table
15	July, 2023	> Updated the What's New table > Updated the DRIVE OS 5.2 to 6.0 Packaging Changes section for TensorRT > Added MCU Communication Coordinator Changes section > Updated the NvMedia Image Processing Pipeline (IPP) Changes Introduced in DRIVE OS 5.1.15.0 section > Added Camera SIPL GPIO and Interrupt Localization Changes Introduced in DRIVE OS 6.0.6.0 section
16	August, 2023	> Added SIPL Event Deprecation section for 6.0.8 > Updated the table, Migrating PKCS#11 Library from 5.2 to 6.0, for 6.0.8 and 6.0.8.1 API changes
17	December, 2023	> Updates to the NvMedia Lens Distortion Correction table > Minor text updates for clarity
18	January, 2024	> Added information about the authentication of camera modules for 6.0.9.1. > Updated the PKCS#11--Implementation Details table with 6.0.9.1 API changes

Version	Date	Description of Change
19	April, 2024	> Updated the Board Support Packages (BSP) API Changes table to add NvThermmon API information > Added API Changes for NVIDIA DRIVE OS VM State Management (NvDVMS)

Table of Contents

List of Tables

Overview

> **Note:** Any services, features, functionality, or enhancements referenced in this document that are not in a GA (general availability) release are subject to change.

This document summarizes the changes you can expect moving from NVIDIA DRIVE® OS 5.1 and 5.2 to 6.0. While NVIDIA DRIVE Xavier™ supports DRIVE OS 5.2, NVIDIA DRIVE Orin™, our next-generation System on a Chip (SoC) for Automotive and Safety, supports DRIVE OS 6.0.

This migration guide is designed to help you plan early to migrate your applications for a seamless experience when you gain access to Orin development and reference production boards starting with 6.0 releases.

The following figure identifies component changes between 5.2 and 6.0.

Table 1.   What's New in NVIDIA DRIVE OS 6

NVIDIA DRIVE OS COMPONENTS	DRIVE OS 5.2	DRIVE OS 6.0	DRIVE OS 6.5
Ubuntu Host Development Environment ---------------------------- **Ubuntu Target Root File System** [1]	18.04	**20.04**	
Linux Kernel[1]	4.14	**5.1**	
Blackberry QNX SDP [2]	7.0.4	**7.1.1**	
Blackberry QNX QOS [2]	2.1	**2.2**	
QCC Toolchain	5.3	**8.3**	
GCC Toolchain	5.4	**9.3**	
C++ Feature Set	14	**17**	
DriveWorks	4	**5**	
CUDA Toolkit	10.2	**11.4**	
NVIDIA UDA CUDA Driver[1] **(x86)**	r450	**r470**	

NVIDIA DRIVE OS COMPONENTS	DRIVE OS 5.2	DRIVE OS 6.0	DRIVE OS 6.5
TensorRT	6.4	**8.x[4]**	
CuDNN	7.6	**8.x[3]**	
Vulkan	1.2		
Wayland	1.17		
Vulcan SC	N/A		**1.0**
QNX Screen		N/A	**7.1.1**
PKCS#11	✓[4]	✓	
HARDWARE	DRIVE OS 5.2	DRIVE OS 6.0	DRIVE OS 6.5
Xavier/Xavier DevKit	✓	**X**	
Orin/Orin Devkit	**X**	✓	
SENSORS			
OnSemi AR0231	✓	-	
OnSemi AR0820			
Sony IMX390			
Sony IMX728	**X**	✓	
Sony IMX623			
OV OV2311			
OTHER			
Packaging	.run files	**Debian/Docker**	
Distribution	NVONLINE	**NGC**	

Notes:

1) Linux only; not available on QNX

2) QNX only; not available on Linux

3) GCC toolchain will be either 9.2 or 9.3

4) Final version number is 6.0.9

With the NVIDIA DRIVE OS 6.0 release, you can expect changes to the following:

> Packaging

> Installation and Tools

> Utilities

> The SDK Structure

> Safety Services

> APIs

DRIVE OS 5.2 to 6.0 Packaging Changes

NVIDIA DRIVE OS 6.0 includes packaging changes from 5.2. This document describes the transition from `.run` files to `.deb` files for Standard and Safety releases.

The following NVIDIA CUDA® package files remain unchanged from 5.2 to 6.0:

> **Note:** The CUDA versions will be updated from 10.2 to 11.x.

> Developer Tools
> • NsightSystems-linux-nda-2021.2.2.3-9b295cc.deb and NVIDIA_Nsight_Graphics_D5Q_NDA_2020.5.20339.deb
> NVIDIA CUDA®
> • cuda-repo-cross-aarch64-qnx-standard-<cuda_version>-local_<cuda_version>.<build_version>_all.deb
> • cuda-repo-cross-qnx-safe-<cuda_version>-local_<cuda_version>.<build_version>_all.deb
> • cuda-repo-minimal-toolkit-<cuda_version>-local_<cuda_version>.<build_version>_amd64.deb
> • cuda-repo-qnx-<cuda_version>-local_<cuda_version>.<build_version>_amd64.deb
> • cuda-repo-ubuntu2004-<cuda_version>-local_<cuda_version>.<build_version>-450.118-1_amd64.deb
> cuDNN
> • cudnn-local-repo-ubuntu{VERSION}_arm64.deb
> • cudnn-local-repo-ubuntu{VERSION}_amd64.deb
> • cudnn-local-repo-cross-aarch64-d6l-{VERSION}_all.deb
> • cudnn-local-repo-cross-aarch64-qnx-{VERSION}.{BUILD}_all.deb
> TensorRT
> • nv-tensorrt-repo-ubuntu2004-{CUDA_VERSION}-trt{VERSION}-x86-host-ga-{BUILD}_amd64.deb
> • nv-tensorrt-repo-ubuntu2004-{CUDA_VERSION}-trt{VERSION}-d6l-target-ga-{BUILD}_arm64.deb

- nv-tensorrt-repo-ubuntu2004-{CUDA_VERSION}-trt{VERSION}-d6l-cross-ga-{BUILD}_amd64.deb
- nv-tensorrt-repo-ubuntu2004-{CUDA_VERSION}-trt{VERSION}-qnx-cross-ga-{BUILD}_amd64.deb
- nv-tensorrt-repo-ubuntu2004-{CUDA_VERSION}-trt{VERSION}-qnx-safe-cross-ga-{BUILD}_1-1_amd64.deb

The following table describes variables used in this document:

Table 2.   List of Variables

Field	Mandatory	DRIVE OS Values
DrivePlatform	Yes	driveos
Flavor	Yes, but applicable only for Master Debian packages	flash, build
OS	Yes	common
DriveType	Yes	safety, standard In the case of "standard", this field will not be populated, and without adjacent dashes
Module	Yes, only for shell and leaf Debian packages	Component name
PDK	Yes, for Master and optional for leaf and shell Debian packages	sdk, pdk
DriveVersion	Yes	{RELEASE}–{GCID} For example, 5.2.x.0-GCID, 6.x.0.0-GCID

Master Debian Packages

> **Note:**
> > NVIDIA DRIVE™ OS Software Development Kit (SDK) is used to develop DRIVE OS applications for deployment on NVIDIA DRIVE AGX™ based hardware platforms. NVIDIA recommends installing from the Master Debian package files for SDKs.
>
> > NVIDIA DRIVE™ OS Platform Development Kit (PDK) is used to adapt NVIDIA DRIVE OS to run on custom hardware based on NVIDIA Automotive SoC. NVIDIA provides the flexibility to install up to the leaf from Master for PDKs.
>
> Not all developers have access to all DRIVE OS components. For more information, contact your NVIDIA representative.

Linux

Table 3.   Master Debian Package Names (Linux)

Master Debian Package File Names	Master Debian Package Names
`nv-driveos-build-{DriveType}-sdk-linux_{DriveVersion}_amd64.deb`	`nv-driveos-build-{DriveType}-sdk-linux-{DriveVersion}`
`nv-driveos-flash-{DriveType}-sdk-linux_{DriveVersion}_amd64.deb`	`nv-driveos-flash-{DriveType}-sdk-linux-{DriveVersion}`

QNX (Standard)

Table 4.   Master Debian Package Names (QNX Standard)

Master Debian Package File Names	Master Debian Package Names
`nv-driveos-build-{DriveType}-sdk-qnx_{DriveVersion}_amd64.deb`	`nv-driveos-build-{DriveType}-sdk-qnx-{DriveVersion}`
`nv-driveos-flash-{DriveType}-sdk-qnx_{DriveVersion}_amd64.deb`	`nv-driveos-flash-{DriveType}-sdk-qnx-{DriveVersion}`
`nv-driveos-build-{DriveType}-debug-sdk-qnx_{DriveVersion}_amd64.deb`	`nv-driveos-build-{DriveType}-debug-sdk-qnx-{DriveVersion}`

Foundation (Standard)

The following table describes the 5.2 to 6.0 Standard Foundation transition:

Table 5.   Standard Foundation .run File to .deb File Map

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
`drive-t186ref-foundation-{DriveVersion}-toolchain.run`	`nv-driveos-foundation-{DriveType}-toolchains_{ToolVersion}_amd64.deb`	`nv-driveos-foundation-{DriveType}-gcc-linaro-aarch64-linux-gnu_{DriveVersion}_amd64.deb`
		`nv-driveos-foundation-{DriveType}-gcc-linaro-arm-linux-gnueabi_{DriveVersion}_amd64.deb`
		`nv-driveos-foundation-{DriveType}-gcc-linaro-arm-linux-gnueabihf_{DriveVersion}_amd64.deb`
`drive-t186ref-foundation-{DriveVersion}-release-sdk.run`	`nv-driveos-foundation-{DriveType}-release-sdk_{DriveVersion}_amd64.deb`	`nv-driveos-foundation-{DriveType}-release-sdk-ist_{DriveVersion}_amd64.deb`
		`nv-driveos-foundation-{DriveType}-release-sdk-platform-config_{DriveVersion}_amd64.deb`
		`nv-driveos-foundation-{DriveType}-release-sdk-virtualization_{DriveVersion}_amd64.deb`
		`nv-driveos-foundation-{DriveType}-release-sdk-core-flash-data_{DriveVersion}_amd64.deb`
		`nv-driveos-foundation-{DriveType}-release-sdk-core-flash-tools_{DriveVersion}_amd64.deb`

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
		`nv-driveos-foundation-{DriveType}-release-sdk-flash-data_{DriveVersion}_amd64.deb`
		`nv-driveos-foundation-{DriveType}-release-sdk-flash-tools_{DriveVersion}_amd64.deb`
`drive-t186ref-foundation-oss-src.run`	N/A	`nv-driveos-foundation-{DriveType}-oss-src_{DriveVersion}_amd64.deb`
`drive-t186ref-foundation-{DriveVersion}-debug-overlay.run`	N/A	`nv-driveos-foundation-{DriveType}-debug-overlay_{DriveVersion}_amd64.deb`
`drive-t186ref-foundation-{DriveVersion}-e3550specific.run`	N/A	None - use e3550 specific .zip files containing FW directly
`drive-t186ref-foundation-{DriveVersion}-p3479specific.run`	N/A	None - use p3479 specific .zip files containing FW directly

Linux

The following table describes the 5.2 to 6.0 Linux transition:

Table 6.   Standard Linux .run File to .deb File Map

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
`drive-t186ref-linux-{DriveVersion}-oss-minimal-sdk.run`	`nv-driveos-linux-oss-minimal-sdk_{DriveVersion}_amd64.deb`	`nv-driveos-linux-oss-minimal-sdk-kernel_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-oss-minimal-sdk-kernel-rt-patches_{DriveVersion}_amd64.deb`

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
		`nv-driveos-linux-oss-minimal-sdk-core_{DriveVersion}_amd64.deb`
`drive-t186ref-linux-{DriveVersion}-oss-src.run`	N/A	`nv-driveos-linux-oss-src_{DriveVersion}_amd64.deb`
`drive-t186ref-linux-{DriveVersion}-initramfs.run`	N/A	`nv-driveos-linux-initramfs_{DriveVersion}_amd64.deb`
`drive-t186ref-linux-{DriveVersion}-nv-minimal-sdk.run`	`nv-driveos-linux-nv-minimal-sdk_{DriveVersion}_amd64.deb`	`nv-driveos-linux-nv-minimal-sdk-samples_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-nv-minimal-sdk-bin-target_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-nv-minimal-sdk-bin-core_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-nv-minimal-sdk-base_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-nv-minimal-sdk-usermode-multimedia_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-nv-minimal-sdk-usermode-compute_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-nv-minimal-sdk-usermode-nvsci_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-nv-minimal-sdk-usermode-graphics_{DriveVersion}_amd64.deb`

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
		`nv-driveos-linux-nv-minimal-sdk-usermode-buildkit_{DriveVersion}_amd64.deb`
`drive-t186ref-linux-{DriveVersion}-dds.run`	`nv-driveos-linux-dds_{DriveVersion}_amd64.deb`	`nv-driveos-linux-dds-install-static_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-dds-isamples_{DriveVersion}_amd64.deb`
		`nv-driveos-linux-dds-core_{DriveVersion}_amd64.deb`
`drive-t186ref-linux-{DriveVersion}-nvros.run`	N/A	`nv-driveos-linux-nvros_{DriveVersion}_amd64.deb`
`drive-t186ref-linux-{DriveVersion}-dpx-patch-v0.0.run`	None - Must be able to cleanly apply new shell/leaf .debs individually for patch updates	`nv-driveos-linux-dpx-patch_{DriveVersion}_amd64.deb`
`drive-t186ref-linux-{DriveVersion}-driveos-core-rfs.run / nvidia-driveos-{Release}-driveos-core-rfs.deb`	N/A	`nv-driveos-linux-driveos-core-rfs_{DriveVersion}_amd64.deb`
`drive-t186ref-linux-{DriveVersion}-driveos-oobe-rfs.run / nvidia-driveos-{Release}-driveos-oobe-rfs.deb`	N/A	`nv-driveos-linux-driveos-oobe-rfs_{DriveVersion}_amd64.deb`
`drive-t186ref-linux-{DriveVersion}-nv-core-rfs.run / nvidia-driveos-{Release}-nv-core-rfs.deb`	N/A	`nv-driveos-linux-nv-core-rfs_{DriveVersion}_amd64.deb`
`nvidia-driveos-{DriveVersion}-copytarget.deb`	N/A	`nv-driveos-linux-{DriveType}-copytarget-{DriveVersion}_amd64.deb`
`nvidia-driveos-{DriveVersion}-buildkit.deb`	N/A	`nv-driveos-linux-{DriveType}-buildkit-{DriveVersion}_amd64.deb`

QNX (Standard)

The following table describes the 5.2 to 6.0 Standard QNX transition:

Table 7.   Standard QNX .run File to .deb File Map

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
`drive-t186ref-qnx-{DriveVersion}-tegra2aurix_updater.run`	N/A	`nv-driveos-qnx-{DriveType}-tegra2aurix-updater_{DriveVersion}_amd64.deb`
`drive-t186ref-qnx-{DriveVersion}-dds.run`	N/A	`nv-driveos-qnx-{DriveType}-dds_{DriveVersion}_amd64.deb`
`drive-t186ref-qnx-<REL>-<GCID>-driver-sdk.run`	`nv-driveos-qnx-driver-sdk_{DriveVersion}_amd64.deb`	`nv-driveos-qnx-{DriveType}-nv-minimal-sdk-base_{DriveVersion}_amd64.deb`
		`nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-multimedia_{DriveVersion}_amd64.deb`
		`nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-compute_{DriveVersion}_amd64.deb`
		`nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-nvsci_{DriveVersion}_amd64.deb`
		`nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-graphics_{DriveVersion}_amd64.deb`
		`nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-buildkit_{DriveVersion}_amd64.deb`
`drive-t186ref-qnx-{DriveVersion}-tensorrt-qnx.run`	N/A	N/A
`drive-t186ref-qnx-{DriveVersion}-tgv.run`	N/A	`nv-driveos-qnx-{DriveType}-tgv_{DriveVersion}_amd64.deb`

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
drive-t186ref-qnx-{DriveVersion}-nvsomeip.run	N/A	nv-driveos-qnx-{DriveType}-nvsomeip_{DriveVersion}_amd64.deb
nvidia-driveos-{DriveVersion}-driveos-qnx-fs.deb	N/A	nv-driveos-qnx-{DriveType}-fs-{DriveVersion}_amd64.deb
nvidia-driveos-{DriveVersion}-copytarget.deb	N/A	nv-driveos-qnx-{DriveType}-copytarget-{DriveVersion}_amd64.deb
nvidia-driveos-{DriveVersion}-buildkit.deb	N/A	nv-driveos-qnx-{DriveType}-buildkit-{DriveVersion}_amd64.deb

QNX (Safety)

The following table describes the 5.2 to 6.0 Safety QNX transition:

Table 8.   Safety QNX .run File to .deb File Map

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
drive-t186ref-qnx-<REL>-<GCID>-driver-sdk.run	nv-driveos-qnx-driver-sdk_{DriveVersion}_amd64.deb	nv-driveos-qnx-{DriveType}-nv-minimal-sdk-base_{DriveVersion}_amd64.deb
		nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-multimedia_{DriveVersion}_amd64.deb
		nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-compute_{DriveVersion}_amd64.deb
		nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-nvsci_{DriveVersion}_amd64.deb
		nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-graphics_{DriveVersion}_amd64.deb

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
		`nv-driveos-qnx-{DriveType}-nv-minimal-sdk-usermode-buildkit_{DriveVersion}_amd64.deb`
`drive-t186ref-qnx-{DriveVersion}-tensorrt-qnx.run`	N/A	N/A
`drive-t186ref-qnx-{DriveVersion}-tgv.run`	N/A	`nv-driveos-qnx-{DriveType}-tgv_{DriveVersion}_amd64.deb`
`drive-t186ref-qnx-{DriveVersion}-debug-overlay.run`	N/A	`nv-driveos-qnx-{DriveType}-debug-overlay_{DriveVersion}_amd64.deb`
`nvidia-driveos-{DriveVersion}-driveos-qnx-safety-fs.deb`	N/A	`nv-driveos-qnx-{DriveType}-fs-{DriveVersion}_amd64.deb`
`nvidia-driveos-{DriveVersion}-driveos-qnx-safety-debug-fs.deb`	N/A	`nv-driveos-qnx-{DriveType}-debug-fs-{DriveVersion}_amd64.deb`
`nvidia-driveos-{DriveVersion}-copytarget.deb`	N/A	`nv-driveos-qnx-{DriveType}-copytarget-{DriveVersion}_amd64.deb`
`nvidia-driveos-{DriveVersion}-buildkit.deb`	N/A	`nv-driveos-qnx-{DriveType}-buildfs-{DriveVersion}_amd64.deb`

Common

The following table describes the 5.2 to 6.0 transition for files common to both Linux and QNX:

Table 9.   Common .run File .deb File Map

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
`nvidia-driveos-{DriveVersion}-copytarget.deb`	N/A	`nv-driveos-common-{DriveType}-copytarget-{Version}_amd64.deb`

Existing Runfile	Shell Debian Package File Name	Leaf Debian Package File Names
`nvidia-driveos-{DriveVersion}-buildkit.deb`	N/A	`nv-driveos-common-{DriveType}-build-fs-{Version}_amd64.deb`

Installation and Tools Changes

Linux targetfs directory to targetfs image

> DRIVE OS Linux defaults to using targetfs image (EXT4 image) while flashing the target.
  - For example, `<NV_WORKSPACE>/drive-linux/filesystem/targetfs.img` will be flashed as the Linux `rootfs` instead of `<NV_WORKSPACE>/drive-linux/filesystem/targetfs/`

> File additions and modifications required to be reflected in the target need to occur by editing the EXT4 image using one of the following ways.
  - Using the Build-FS tool (Recommended). For additional information, refer to the following topics in the NVIDIA DRIVE OS 6.0 Linux SDK Developer Guide.
    - [Editing NVIDIA Build-FS CONFIG](#)
    - [Executing NVIDIA Build-FS With the Updated CONFIG](#)
    - [Flashing the Customized Target Filesystem](#)
  - Editing of `<NV_WORKSPACE>/drive-linux/filesystem/targetfs.img` :
    1. `mkdir /tmp/mount`
    2. `sudo mount <NV_WORKSPACE>/drive-linux/filesystem/targetfs.img /tmp/mount`
    3. Make minor edits in `/tmp/mount`
    4. `sudo umount /tmp/mount`

    This is only possible for small edits as the targetfs.img free space is limited

> Debian to install the filesystem in DRIVE OS Linux has been updated from:
  ```
  nv-driveos-linux-<fs_variant>-tar-extract-<release>-<gcid>_<release>-
  <gcid>_amd64.deb to nv-driveos-linux-<fs_variant>-<release>-<gcid>_<release>-
  <gcid>_amd64.deb
  ```

  Where

  ```
  <fs_variant> is one from (driveos-core-ubuntu-20.04-rfs, driveos-oobe-ubuntu-
  20.04-rfs, driveos-oobe-desktop-ubuntu-20.04-rfs),
  ```
  `<release>` is the DRIVE OS RELEASE

  `<gcid>` is the DRIVE OS GCID

> To ensure backwards compatibility and allow a smooth transition, DRIVE OS Linux still creates the `<NV_WORKSPACE>/drive-linux/filesystem/targetfs/` directory when the new Debian package is installed.

- If `<NV_WORKSPACE>/drive-linux/filesystem/targetfs/` is not available in your installation, ensure your system supports loop mounting ([Ubuntu Manpage: losetup - set up and control loop devices](#)) and run the following commands for manual mount of Ext4 image:

```
mkdir -p /tmp/mount <NV_WORKSPACE>/drive-linux/filesystem/targetfs/
sudo mount -t ext4 <NV_WORKSPACE>/drive-linux/filesystem/targetfs.img /tmp/mount
sudo cp -a /tmp/mount/. <NV_WORKSPACE>/drive-linux/filesystem/targetfs/
```

Optional Debian packages on installation will update both `<NV_WORKSPACE>/drive-linux/filesystem/targetfs.img` and `<NV_WORKSPACE>/drive-linux/filesystem/targetfs/` directory.

> The directory `<NV_WORKSPACE>/drive-linux/filesystem/targetfs/` is supported in DRIVE OS for the entire DRIVE OS 6.0.x release cycle.

> To revert PCT to pick up `rootfs` contents from `<NV_WORKSPACE>/drive-linux/filesystem/targetfs/` similar to behavior in previous releases:

- After bind partitions (such as `make -f Makefile.bind`), run `<NV_WORKSPACE>/drive-foundation/tools/misc/use_rootfs_img.sh -r`

Toolchain Upgrade

Toolchain in DRIVE OS cross compilation is upgrading from GCC 7.3.1 to GCC 9.3.0, and the Toolchain vendor is changing from Linaro to Bootlin.

Table 10.   Legacy Toolchain and Replacement Toolchain

Legacy Toolchain in DRIVE OS 5.2	Replacement Toolchain in DRIVE OS 6.0
`<INSTALL_DIR>/toolchains/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu`	`<INSTALL_DIR>/toolchains/aarch64--glibc--stable-2020.08-1`
`<INSTALL_DIR>/toolchains/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabi`	`<INSTALL_DIR>/toolchains/armv5-eabi--glibc--stable-2020.08-1`
`<INSTALL_DIR>/toolchains/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf`	`<INSTALL_DIR>/toolchains/armv7-eabihf--glibc--stable-2020.08-1`

You can install the new toolchains in DRIVE OS two different ways.

> **Runfile Installation**: `drive-<NVPLATFOTM>-foundation-<VERSION>-toolchain.run` installs the new toolchain binaries.

> **Debian Installation:** Three new Debian packages are available to install the individual toolchains:
>   - `nv-driveos-foundation-gcc-bootlin-gcc9.3-aarch64--glibc--stable-2020.08-1_9.3.0_amd64.deb`
>   - `nv-driveos-foundation-gcc-bootlin-gcc9.3-armv5-eabi--glibc--stable-2020.08-1_9.3.0_amd64.deb`
>   - `nv-driveos-foundation-gcc-bootlin-gcc9.3-armv7-eabihf--glibc--stable-2020.08-1_9.3.0_amd64.deb`
>
>   These can be installed using the APT package manager and by providing the `NV_WORKSPACE` value when prompted. (Refer to the installation documentation for more details)
> Bind partitions using `Makefile.bind` on the user side require the updated aarch64 toolchain (`aarch64--glibc--stable-2020.08-1/`)
> Rebuild of QB and Kernel from DRIVE OS require the updated aarch64 toolchain (`aarch64--glibc--stable-2020.08-1/`)
> Rebuild the entire software stack with the updated toolchain instead of mixing toolchain versions.
> Shared libraries (.so) generated with older compilers are theoretically compatible and linkable with the current toolchain.
>   - However, shared libraries (.so) generated with new toolchain can be incompatible when linked with the older toolchain due to references to newer library APIs in the '.so'

Ubuntu Upgrade

The Ubuntu distribution version was upgraded from 18.04 to 20.04.

You have two different installation options.

> **Runfile Installation:**
>   - Select either `driveos-core-rfs` or `driveos-oobe-rfs`.
>   - Install the run file depending on core/oobe RFS respectively:
>   - `drive-<NVPLATFORM>-linux-<VERSION>-driveos-oobe-ubuntu-20.04-rfs.run`
>   - `drive-<NVPLATFORM>-linux-<VERSION>-driveos-core-ubuntu-20.04-rfs.run.`
> **Debian Installation:**
>   - NVIDIA provides two Debian packages.
>   - `nvidia-driveos-<VERSION>-driveos-core-ubuntu-20.04-rfs.deb`
>   - `nvidia-driveos-<VERSION>-driveos-oobe-ubuntu-20.04-rfs.deb`
>   - The Debian packages are installable side-by-side. Filesystem images of core/oobe, respectively, are as follows:
>   - `/opt/nvidia/driveos/6.0.0.0/filesystems/driveos-core-ubuntu-20.04-rfs/driveos-core-ubuntu-20.04-rfs.img`

- `/opt/nvidia/driveos/6.0.0.0/filesystems/driveos-oobe-ubuntu-20.04-rfs/driveos-oobe-ubuntu-20.04-rfs.img`

Ubuntu 20.04, no longer provides the package python. You must install python2 or python3 and execute it as such.

> sudo apt install python2 python3

> python2

> python3

Some pip plugins are retired and will not work with python2. For developing new applications, use python3 instead of python2.

Changes to Files in targetfs

NITO File Path Changes in 6.0.6

Camera NITO files moved from `/opt/nvidia/nvmedia/nit/` to `/usr/bin/camera`.

MCU Communication Coordinator Changes

Bootchain command responses have a data length of 100 bytes in 6.0 vs 9 bytes in 5.2. The response layout is the same, but there are trailing zeroes in 6.0.x.

The number of bootchains is higher in Orin, which affects the content of frames that include the bootchain number.

There is no Tegra reboot command in 6.0.x.

Utilities

Migration QNX IOLauncher config File to Device Tree

For integration with DRIVEOS State Management, all processes must start from the startup command device tree.

Device Tree property details are described in the "Starting process from the device tree" section of IOLauncher utility manual. For more information, refer to the *IO Launcher* topic in the NVIDIA DRIVE OS 6.0 PDK for QNX or Linux.

Each command line in the IOLauncher config file migrates as `cmd` property of each command node in the startup command device node. `cmd` property has the same command line string as the source config file.

IOLauncher executes the commands sequentially, as listed in `start_seq` property. The order of command nodes does not affect the execution sequence.

`critical_process` property is `yes` if the command has the `--critical` iolauncher option. Otherwise, the property is `no`.

`heartbeat` and `oneshot` are reserved properties for the future. They have no impact on any behavior.

`sc7` property has a corresponding value as described in the IOlauncher utility manual.

Table 11.   Startup Command DT Files

DT Files	Applicable to
tegra234-startupcmds-gos0-safety.dtsi	QNX Safety GOS0 VM
tegra234-startupcmds-gos0.dtsi	QNX Standard GOS0 VM
tegra234-startupcmds-gos1-safety.dtsi	QNX Safety GOS1 VM
tegra234-startupcmds-gos1.dtsi	QNX Standard GOS1 VM

SDK Structure

DRIVE OS 6.0 presents an improved directory structure and file layout over previous versions. The SDK contents are presented in a more organized fashion, making files more discoverable. Related files for a particular area of functionality are now localized to a single location.

New Top-Level Directories

New top-level directories are used to localize all related files to a single directory. In previous versions of DRIVE OS, these contents were scattered across multiple locations.

> **filesystem/** contains all files related to the population of the target filesystem for a Guest OS.
> **tools/** contains all tools, scripts, and utilities intended to run on the host.
> **kernel/** contains all files related to the Linux kernel.
> **platform-config/** contains all files related to platform configuration.
> **firmware/** contains all files that can be categorized as firmware.

There are some cases where specific files might fall into more than one area of functionality. In those cases, the files were placed into the most appropriate location.

t186ref Removal from the SDK Installation Directory and File Names

The NVIDIA DRIVE Xavier SoC chip is no longer supported in DRIVE OS 6.0 SDK. Furthermore, the DRIVE OS SDK is designed to support multiple Tegra chips, so no one chip should be in the SDK name.

This change is expected to have a large impact on customers who used prior releases of DRIVE OS. The runfile names previously containing drive-t186ref-foundation, drive-t186ref-linux and drive-t186ref-qnx will now contain drive-foundation, drive-linux, and drive-qnx. Similarly, the top-level installation directories, previously drive-t186ref-foundation/, drive-t186ref-linux/, and drive-t186ref-qnx/, are now drive-foundation/, drive-linux/, and drive-qnx/. Customers who wrote installation scripts, test scripts,

Makefiles, documentation pages, or other files referencing the old SDK names will need to make modifications to use the new names

SDK File Locations in 5.2 and 6.0

The following table describes how file locations have changed between 5.2 and 6.0:

Table 12.　　SDK File Locations in 5.2 and 6.0

	DRIVE OS 5.2	DRIVE OS 6.0
File-system	5.2	6.0
	`drive-t186ref-linux/`	
	`bin-target/*`	`filesystem/contents/bin/*`
	`data/conf/*.conf`	`filesystem/contents/config/x11/`
	`kernel-rt_patches/yocto-tegra-initramfs-rootfs.img`	`filesystem/initramfs.cpio`
	`modules/*.rules`	`filesystem/contents/config/startup/`
	`modules/scripts/*`	`filesystem/contents/config/startup/scripts/`
	`OOBE/lib/systemd/system/drive-setup.service`	`filesystem/contents/config/OOBE/`
	`targetfs-pkgs/filesystem.tar.bz2`	`filesystem/targetfs-pkgs/filesystem.tar.bz2`
	`target-images/debians/*.deb`	`filesystem/contents/debians/`
	`targetfs-images/(*.img and MANIFEST.json)`	`filesystem/targetfs-images/(*.img and MANIFEST.json)`
	`targetfs/`	`filesystem/targetfs/`
	`utils/nvpmodel/*`	`filesystem/contents/ (various subdirectories)`
	`utils/scripts/copytarget/manifest/*.yaml`	`filesystem/copytarget/manifest/*.yaml`
	`vulkan/icd.d/nvidia_icd.json`	`filesystem/contents/config/vulkan/`
	`rootfs-licenses.txt`	`filesystem/rootfs-licenses.txt`
	`drive-t186ref-qnx/ (standard)`	
	`bin-target/*`	`filesystem/contents/bin/*`
	`targetfs-images/*`	`filesystem/targetfs-images/*`
	`build-fs-configs/driveos-qnx-fs.CONFIG.json`	`filesystem/build-fs/configs/driveos-qnx-fs.CONFIG.json`
	`qnx_create_targetfs*.sh`	`filesystem/qnx_create_targetfs*.sh`
	`rootfs-licenses.txt`	`filesystem/rootfs-licenses.txt`

	DRIVE OS 5.2	DRIVE OS 6.0
	`targetfs/`	`filesystem/targetfs/`
	`utils/scripts/copytarget/manifest/*.yaml`	`filesystem/copytarget/manifest/*.yaml`
	`vulkan/icd.d/nvidia_icd.json`	`filesystem/contents/config/vulkan/`
	`drive-t186ref-qnx/ (safety)`	
	`bin-target/*`	`filesystem/contents/bin/*`
	`targetfs.build_file`	`filesystem/targetfs.build_file`
	`qnx_targetfs.img`	`filesystem/qnx_targetfs.img`
	`targetfs-images/*`	`filesystem/targetfs-images/*`
	`build-fs-configs/driveos-qnx-safety-fs.CONFIG.json`	`filesystem/build-fs/configs/driveos-qnx-safety-fs.CONFIG.json`
	`qnx_create_targetfs_safety*.sh`	`filesystem/qnx_create_targetfs_safety*.sh`
	`rootfs-licenses.txt`	`filesystem/rootfs-licenses.txt`
	`targetfs/`	`filesystem/targetfs/`
	`utils/scripts/copytarget/manifest/*.yaml`	`filesystem/copytarget/manifest/*.yaml`
Tools	5.2	6.0
	`drive-t186ref-foundation/ (standard)`	
	`firmware/src/trusted_os/ta-dev/tools/eks_gen/*`	`tools/security/eks_gen/*`
	`Makefile.bind`	`make/Makefile.bind`
	`security/trusted-os/scripts/tools/nv_pkcs11_keywrap/*`	`tools/security/pkcs11/keywrap/*`
	`security/trusted-os/scripts/tools/tos_gen/*`	`tools/security/tos_gen/*`
	`tools/host/dtc/dtc`	`tools/device-tree/dtc`
	`tools/host/delnode*`	`tools/device-tree/delnode*`
	`tools/host/flashtools/*`	`tools/flashtools/*`
	`tools/host/ist/*`	`tools/ist/*`
	`tools/host/*muxer/*`	`tools/muxer/*`
	`tools/host/use_rootfs_img.sh`	`tools/misc/`
	`virtualization/tools/eventlib/*`	`tools/perf/eventlib/*`
	`virtualization/tools/pctdump_x86/*`	`tools/pctdump/*`
	`drive-t186ref-foundation/ (safety)`	

	DRIVE OS 5.2	DRIVE OS 6.0
	`Makefile.bind`	`make/Makefile.bind`
	`security/trusted-os/scripts/tools/nv_pkcs11_keywrap/*`	`tools/security/pkcs11/keywrap/*`
	`security/trusted-os/scripts/tools/tos_gen/*`	`tools/security/tos_gen/*`
	`tools/host/dtc/dtc`	`tools/device-tree/dtc`
	`tools/host/delnode*`	`tools/device-tree/*`
	`tools/host/flashtools/*`	`tools/flashtools/*`
	`tools/host/ist/*`	`tools/ist/*`
	`drive-t186ref-linux/`	
	`utils/nv_edidgenerate`	`tools/nv_edidgenerate/nv_edidgenerate`
	`utils/openGL-tools/glslc`	`tools/openGL-tools/glslc`
	`utils/optin_fuse`	`tools/optin_fuse/optin_fuse`
	`drive-t186ref-qnx/ (standard)`	
	`utils/asid_hash_calculator`	`tools/asid_hash/asid_hash_calculator`
	`utils/emc_log`	`tools/emc_log/emc_log`
	`utils/fpdlink_init/*`	`tools/fpdlink_init/*`
	`utils/host/laptsa-imp`	`tools/laptsa-imp/laptsa-imp`
	`utils/memuage`	`tools/memusage/memusage`
	`utils/nvcanrtc/nvcanrtc`	`tools/nvcanrtc/nvcanrtc`
	`utils/openGL-tools/glslc`	`tools/openGL-tools/glslc`
	`utils/thermal_utils/*`	`tools/thermal_utils/*`
	`drive-t186ref-qnx/ (safety)`	
	`utils/host/laptsa-imp`	`tools/laptsa-imp/laptsa-imp`
Linux Kernel	5.2	6.0
	`drive-t186ref-linux/`	
	`kernel-rt-patches/{Image,System.map,vmlinux}`	`kernel/preempt_rt/images/{Image,System.map,vmlinux}`
	`kernel-rt-patches/modules/*`	`kernel/preempt_rt/modules/*`
	`drive-oss-src/`	

	`*`	`kernel/source/oss_src/kernel`
		`kernel/source/oss_src/nvgpu`
		`kernel/source/oss_src/nvidia`
		`kernel/source/oss_src/nvlink`

	`hardware/`	

	DRIVE OS 5.2	DRIVE OS 6.0
	`*`	`kernel/source/hardware`
Platform Config	5.2	6.0
	Xavier (t19x) platform configuration files across multiple directories will be removed from DRIVE OS 6.0.	Orin (t23x) platform configuration files are localized to the platform-config/ directory in each SDK.
Firmware	5.2	6.0
	`drive-t186ref-foundation/ (Standard and Safety)`	
	`firmwares/*`	`firmware/*`
	`drive-t186ref-linux/`	
	`lib-target/<firmware files>`	`firmware/<firmware files>`
	`drive-t186ref-qnx/ (Standard and Safety)`	
	`lib-target/<firmware files>`	`firmware/<firmware files>`
	`nvidia-bsp/aarch64le/usr/lib/<firmware files>`	`firmware/<firmware files>`
t186ref Removal	5.2	6.0
	`drive-t186ref-foundation/ (Standard)`	`drive-foundation/`
	`drive-t186ref-foundation/ (Safety)`	`drive-foundation-safety/`
	`drive-t186ref-linux/`	`drive-linux/`
	`drive-t186ref-qnx/ (Standard)`	`drive-qnx/`
	`drive-t186ref-qnx/ (Safety)`	`drive-qnx-safety/`
Other	`5.2`	`6.0`
	`drive-t186ref-linux/`	
	`oss/X11/*`	`filesystem/targetfs/ from` (all X11 libs and headers are present in appropriate locations)
	`oss/{ssl,usb}/`	`filesystem/targetfs/` (libs only in appropriate locations, headers deprecated)

API Changes in DRIVE OS 6.0

Overview

NVIDIA DRIVE OS 6.0 introduces API changes to the modules shown in the following figure:

DRIVE PLATFORM APIs & SDKs
What's New

Middleware & Platform APIs	Workstation / Cloud (x86)		Embedded Target (Tegra)			
	Linux		Linux		QNX OS for Safety (QOS)	
	Standard	Safety Proxy[1]	Standard	Safety Proxy[1]	Standard[2]	Safety[3]
DriveWorks[4]						
TensorRT			✔			
CUDA						
NvStreams		Common DRIVE Platform APIs for Production Deployment				
NvMedia						
NvMedia SIPL		✖				
NvMedia DLA						
Vulkan SC					Coming in DRIVE OS 6.1	
Vulkan						
cuDNN						
OpenGL ES	✔	✖	✔	✖	✔	✖
EGL						
EGL Streams						
OpenGL						

Notes:
1) Approximation of safety on non-safety platforms for development.
2) For development only, not for use in production.
3) Refer to the DRIVE OS Safety Manual and QNX Safety Manuals for safety context definitions.
4) Refer to DRIVE platform roadmap for scope of DriveWorks Safety Proxy and Safety Builds

> **Note:** Safety Proxy APIs are identical to Safety APIs on QNX and supported in the Standard build.

NvMedia API Changes

Summary of the NvMedia API Timeline

Release 6.0.2.0

NvMediaImage, NvMediaImage-based, and NvSciBuf APIs are available.

Release 6.0.3.0:

QNX Safety no longer supports NvMediaImage APIs and NvMediaImage-based APIs.
QNX Standard and Linux use NvMediaImage APIs and NvMediaImage-based APIs.

Release 6.0.4.0:

NvMediaImage APIs and NvMediaImage-based APIs are deprecated.

Release 6.0.7.0:

NvMediaEncodeQuality is deprecated in 6.0.7 and replaced by NvMediaEncPreset, which
is supported in 6.0.6.0 and later releases.

Release 6.0.9.1

Authentication of all camera modules is enforced when FUSE_SECURITY_MODE_0 (FSKP
XML tag: "SecurityMode") is 1. DRIVE OS SIPL core refuses an application request to
start a camera streaming for any module whose authentication cannot finish
successfully. For additional information, refer to "Platform Camera Configuration" in the
NVIDIA DRIVE OS QNX SDK Developer Guide. For information regarding camera module
sensor authentication support, see "Adding Authentication Support for a New Camera
Sensor" in the *NVIDIA DRIVE OS QNX PDK Developer Guide*.

> **Note:** For releases with both NvMediaImage and NvSciBuf APIs, the NvSciBuf based
> APIs and test applications are packaged in the nvmedia_6x directories.
>
> When using NvSciBuf APIs, include the nvmedia_6x directory first followed by
> the default include directories

NvMediaEncodeQuality Deprecation

`NvMediaEncodeQuality` is deprecated in 6.0.7, which requires migrating to `NvMediaEncPreset`. Both APIs are defined in `nvmedia_common_encode.h`, which is part of the IEP (Image Encode Processing) API.

To enable NvMediaEncPreset in 6.0.6 (using the H264 codec as an example):

`NvMediaEncodeConfigH264::enableNewPreset = true`

`NvMediaEncodeConfigH264::NvMediaEncPreset = <value>`

`NvMediaEncodeConfigH264::NvMediaEncodeQuality`, if set is ignored

To enable NvMediaEncPreset for 6.0.7.0 and later releases:

`NvMediaEncodeConfigH264::NvMediaEncPreset = <value>`

Table 13.　　　Mapping between NvMediaEncodeQuality and NvMediaEncPreset

Codec	Nvmedia Previous Name (NvMediaEncodeQuality)	Nvmedia New Preset (NvMediaEncPreset)
H264	NVMEDIA_ENCODE_QUALITY_L0	NVMEDIA_ENC_PRESET_UHP
	NVMEDIA_ENCODE_QUALITY_L1	NVMEDIA_ENC_PRESET_HP
	NVMEDIA_ENCODE_QUALITY_L2	NVMEDIA_ENC_PRESET_HQ
H265	NVMEDIA_ENCODE_QUALITY_L0 + (Feature) NVMEDIA_ENCODE_CONFIG_H265_ ENABLE_ULTRA_FAST_ENCODE	NVMEDIA_ENC_PRESET_UHP
	NVMEDIA_ENCODE_QUALITY_L0	NVMEDIA_ENC_PRESET_HP
	NVMEDIA_ENCODE_QUALITY_L1	NVMEDIA_ENC_PRESET_HQ
AV1	NVMEDIA_ENCODE_QUALITY_L2	NVMEDIA_ENC_PRESET_UHP
	NVMEDIA_ENCODE_QUALITY_L1	NVMEDIA_ENC_PRESET_HP

Table 14.   Support Matrix for NvMedia Multimedia APIs

Release	Feature	6.0.0.0	6.0.1.0	6.0.2.0	6.0.3.0	6.0.4.0	Ref.
NvMediaImage based NvMediaImage Decoder (nvmedia_imgdec.h)	Decode via NvDEC	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated	Table 15
NvMedia Video Decode API (nvmedia_viddec.h)		Supported	Supported	Supported	Supported	Deprecated	
NvSciBuf based NvMediaDecoder API (nvmedia_ide.h[1])		N/A	N/A	Supported	Supported	Supported	
NvMedia Video Encode API (nvmedia_vep.h)	Encode via NvENC	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated	Table 16
NvMediaImage based NvMediaIEP API (nvmedia_iep.h, nvmedia_iep_nvscisync.h)		Supported	Supported	Supported	Supported	Deprecated	
NvSciBuf based NvMediaIEP API (nvmedia_iep.h[1])		N/A	N/A	Supported	Supported	Supported	
NvMediaImage based NvMediaIOFST API (nvmedia_iofst.h)	Optical Flow/ Stereo via OFA	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated	Table 17 Table 18
NvMediaImage based NvMediaIOFA API (nvmedia_ofa.h)		Supported	Supported	Supported	Supported	Deprecated	
NvSciBuf based NvMediaIOFA API (nvmedia_iofa.h[1])		N/A	N/A	Supported	Supported	Supported	

NvMediaImage based NvMediaIJPD API (nvmedia_ijpd.h)	JPEG Decode via NvJPEG	Supported	Supported	Supported	Supported	Deprecated	Table 19
NvSciBuf based NvMediaIJPD API (nvmedia_ijpd.h[1])		N/A	N/A	Supported	Supported	Supported	
NvMediaImage base NvMediaIJPE API (nvmedia_ijpe.h)	JPEG Encode via NvJPEG	Supported	Supported	Supported	Supported	Deprecated	Table 20
NvSciBuf based NvMediaIJPE API (nvmedia_ijpe.h[1])		N/A	N/A	Supported	Supported	Supported	

Table 15.   NvMedia Video Decoder

Module: NvMedia Video Decoder	
Header File and API	Comments

Header File:
 	
`nvmedia_viddec.h`
 	
Old Name:
 	
`NvMediaVideoDecoder *`
	NvMedia Video Decoder APIs use the NVDEC HW engine on DRIVE OS 5.2 to decode bitstreams compressed using standard video compression algorithms such as H.264, H.265, VP9, VP8, and MPEG2. These APIs will be deprecated in DRIVE OS 6.0.4.0 release
	
	NvSciBuf-based NvMedia Decoder API will be introduced as a part of the DRIVE OS 6.0.2.0 release which will supersede the existing NvMedia Video Decoder APIs

Old API:
 	
`NvMediaVideoDecoder *`
`NvMediaVideoDecoderCreateEx(`
`    const NvMediaDevice *device,`
	> `NvMediaDevice` will be deprecated and the argument will be removed from the API
	> Additionally the API name has changed

---

[1] NvSciBuf-based APIs are introduced in DRIVE OS 6.0.2.0 separately from the NvMediaImage-based APIs

Module: NvMedia Video Decoder

<table>
<tr>
<td>

```
    NvMediaVideoCodec codec,
    uint16_t width,
    uint16_t height,
    uint16_t maxReferences,
    uint64_t  maxBitstreamSize,
    uint8_t inputBuffering,
    uint32_t flags,
    NvMediaDecoderInstanceId
instanceId
);
```

</td>
<td>

> `NvMediaVideoCodec` will be updated to support AV1 decoding only from T234 and further chips

Header File:

```
nvmedia_ide.h
```

New API:

```
NvMediaIDE *
NvMediaIDECreate(
    NvMediaVideoCodec codec,
    uint16_t width,
    uint16_t height,
    uint16_t maxReferences,
    uint64_t maxBitstreamSize,
    uint8_t inputBuffering,
    uint32_t flags,
    NvMediaDecoderInstanceId
instanceId
);
```

</td>
</tr>
<tr>
<td>

Old API:

```
NvMediaStatus
NvMediaVideoDecoderRenderEx(
    const NvMediaVideoDecoder
*decoder,
    const NvMediaVideoSurface
*target,
    const NvMediaPictureInfo
*pictureInfo,
    const void *encryptParams,
    uint32_t numBitstreamBuffers,
    const NvMediaBitstreamBuffer
*bitstreams,
    NvMediaVideoDecodeStats
*FrameStatsDump,
    NvMediaDecoderInstanceId
instanceId
)
```

</td>
<td>

> `NvMediaImage` will be deprecated and replaced by `NvSciBufObj`
> Additionally, the API name might change

Header File:

nvmedia_ide.h

New API:

```
NvMediaStatus
NvMediaIDEDecoderRender(
    const NvMediaIDE *decoder,
    NvSciBufObj target,
    const NvMediaPictureInfo
*pictureInfo,
    const void *encryptParams,
    uint32_t numBitstreamBuffers,
    const NvMediaBitstreamBuffer
*bitstreams,
    NvMediaIDEFrameStats
*FrameStatsDump,
    NvMediaDecoderInstanceId
instanceId
);
```

</td>
</tr>
</table>

Module: NvMedia Video Decoder	

Old API:
 	
N/A
	NvMediaIDEFillNvSciBufAttrList is a new API introduced in DRIVE OS 6.0 to fill IDE-specific attributes in the NvSciBufAttrList. The NvSciBufAttrList is used during creation of NvSciBufObj. This is a mandatory API
	
	Header File:
	
	    nvmedia_ide.h
	
	NEW API:
	
	<pre>NvMediaStatus
	NvMediaIDEFillNvSciBufAttrList(
	    NvMediaDecoderInstanceId
	instanceId,
	    NvSciBufAttrList attrlist
	);</pre>

Old API:
 	
N/A
	NvMediaIDEFillNvSciSyncAttrList is a new API introduced in DRIVE OS 6.0 to fill IDE-specific Sync attributes in the NvSciBufAttrList. like engindID. This is a mandatory API
	
	Header File:
	
	    nvmedia_ide.h
	New API:
	<pre>NvMediaStatus
	NvMediaIDEFillNvSciSyncAttrList(
	    const NvMediaIDE
	*decoder,
	    NvSciSyncAttrList
	attrlist,
	    NvMediaNvSciSyncClientType
	clienttype
	);</pre>

OLD API: N/A	In 5.2 these APIs are optional. In DRIVE OS 6.0 these APIs are mandatory if IDE needs to be used in a pipeline with other engine APIs In DRIVE OS 6.0.2.0 these APIs will be supported for fence synchronization between Driver and Hardware

Module: NvMedia Video Decoder	

	Header File:
	
	    `nvmedia_ide.h`
	
	New API:
	
	    `# NvMediaIDENvSciSync APIs`
	    `NvMediaIDEFillNvSciSyncAttrList`
	    `NvMediaIDERegisterNvSciSyncObj`
	    `NvMediaIDEUnregisterNvSciSyncObj`
	    `NvMediaIDESetNvSciSyncObjforEOF`
	    `NvMediaIDEInsertPreNvSciSyncFence`
	    `NvMediaIDEGetEOFNvSciSyncFence`

Table 16.   NvMedia IEP - NvMediaImage based to NvSciBuf based

Module: NvMedia IEP	
Header File and API	**Comments**

Header File:
 	
    `nvmedia_iep.h`
    `nvmedia_iep_nvscisync.h`
 	
API:
 	
`NvMediaIEP*`
	The existing NvMediaImage-based NvMediaIEP APIs are supported until the DRIVE OS 6.0.4.0 release and will be superseded by the NvSciBuf based NvMediaIEP APIs. The NvSciBuf based APIs will be introduced in the DRIVE OS 6.0.2.0 release
	
	> The changes between the existing `NvMediaImage`-based -APIs and the proposed `NvSciBuf`-based NvMediaIEP APIs are explained in the following rows

Header File:
`nvmedia_iep.h`
 	
NvMediaImage based API:
 	
`NvMediaIEP *`
`NvMediaIEPCreate(`
    `const NvMediaDevice *device,`
    `NvMediaIEPType encodeType,`
    `const void *initParams,`
    `NvMediaSurfaceType inputFormat,`
    `uint8_t maxInputBuffering,`
    `uint8_t maxOutputBuffering,`
    `NvMediaEncoderInstanceId`
`instanceId`
`);`
	`NvMediaDevice` will be deprecated and the argument will be removed from the API
	
	`NvMediaSurfaceType` will be replaced by `NvSciBufAttrList`, which will describe the properties of the surface in terms of NvSciBuf Attributes
	
	`NvMediaIEPType` will be updated to support AV1 encoding
	
	`maxInputBuffering` will be deprecated and the argument will be removed from the API
	
	    `NvSciBuf based API:`
	    `NvMediaIEP *`
	    `NvMediaIEPCreate(`
	        `NvMediaIEPType encodeType,`
	        `const void *initParams,`
	        `NvSciBufAttrList bufAttrList,`
	        `uint8_t maxBuffering,`

Module: NvMedia IEP	
Header File and API	**Comments**
| | ```
                      NvMediaEncoderInstanceId
    instanceId
    );
``` |
Header File:
 	
    nvmedia_iep.h
 	
NvMediaImage based API:
    NvMediaStatus
    NvMediaIEPFeedFrame(
        const NvMediaIEP *encoder,
        const NvMediaImage *frame,
        NvMediaRect *sourceRect,
        const void *picParams,
        NvMediaEncoderInstanceId
    instanceId
    );
	> NvMediaImage will be deprecated and replaced by NvSciBufObj
	> NvMediaRect will no longer be supported, and the argument will be removed from the API
	
	NvSciBuf based API:
	
	    NvMediaStatus
	    NvMediaIEPFeedFrame(
	        const NvMediaIEP *encoder,
	        NvSciBufObj bufObj,
	        const void *picParams,
	        NvMediaEncoderInstanceId
	    instanceId
	    );

Header File:
nvmedia_iep.h
 	
NvMediaImage based API:
 	
    NvMediaStatus
    NvMediaIEPImageRegister(
        const NvMediaIEP *encoder,
        const NvMediaImage *image,
        NvMediaAccessMode accessMode
    );
	> NvSciBufObj replaces NvMediaImage, which is deprecated with 6.0
	> accessMode is deprecated and removed
	NvSciBuf based API:
	
	    NvMediaStatus
	    NvMediaIEPUnregisterNvSciBufObj(
	        const NvMediaIEP *encoder,
	        NvSciBufObj bufObj
	    );

Header File:
 	
    nvmedia_iep.h
 	
NvMediaImage based API:
 	
    NvMediaStatus
    NvMediaIEPImageUnRegister(
        const NvMediaIEP *encoder,
        const NvMediaImage *image
    );
	NvSciBufObj replaces NvMediaImage, which is deprecated with 6.0
	
	NvSciBuf based API:
	
	    NvMediaStatus
	    NvMediaIEPUnregisterNvSciBufObj(
	        const NvMediaIEP *encoder,
	        NvSciBufObj bufObj
	    );

Header File: nvmedia_iep.h	API renamed to NvMediaIEPGetBits

Module: NvMedia IEP	
Header File and API	**Comments**
NvMediaImage based API: ```NvMediaStatus` `NvMediaIEPGetBitsEx(` ` const NvMediaIEP *encoder,` ` uint32_t *numBytes,` ` uint32_t numBitstreamBuffers,` ` const NvMediaBitstreamBuffer` `*bitstreams,` ` void *extradata` `);```	**NvSciBuf based API:** ```NvMediaStatus` `NvMediaIEPGetBits(` ` const NvMediaIEP *encoder,` ` uint32_t *numBytes,` ` uint32_t numBitstreamBuffers,` ` const NvMediaBitstreamBuffer` `*bitstreams,` ` void *extradata` `);```
Header File: `    N/A` **NvSciBuf based API:** ```NvMediaStatus` `NvMediaIEPFillNvSciBufAttrList(` ` NvMediaEncoderInstanceId` `instanceId,` ` NvSciBufAttrList` `attrlist` `);```	`NvMediaIEPFillNvSciBufAttrList` is a new API introduced in DRIVE OS 6.0 to fill IEP-specific attributes in the `NvSciBufAttrList`. The `NvSciBufAttrList` is used during creation of `NvSciBufObj`. This is a mandatory API, which needs to be called to populate `NvSciBufAttrList` that would eventually get used during `NvSciBufObj` allocation.

Header File:
 	
`    nvmedia_iep_nvscisync.h`
 	
NvMediaImage based API:
`    NvMediaIEPNvSciSyncGetVersion`
`    NvMediaIEPFillNvSciSyncAttrList`
`    NvMediaIEPRegisterNvSciSyncObj`
`    NvMediaIEPUnregisterNvSciSyncObj`
`    NvMediaIEPSetNvSciSyncObjforEOF`
`    NvMediaIEPInsertPreNvSciSyncFence`
`    NvMediaIEPGetEOFNvSciSyncFence`
`    NvMediaIEPSetNvSciSyncObjforSOF`
`    NvMediaIEPGetSOFNvSciSyncFence`
	In 5.2 these APIs are optional. In DRIVE OS 6.0 these APIs are mandatory if IEP needs to be used in a pipeline with other engine APIs

Header File:
 	
`    nvmedia_iep_nvscisync.h`
 	
NvMediaImage based API:
`    NvMediaIEPSetNvSciSyncObjforSOF`
`    NvMediaIEPGetSOFNvSciSyncFence`
	In DRIVE OS 5.2 these APIs were not supported, but they will be supported in 6.0

Module: NvMedia IEP	
Header File and API	**Comments**

Header File:
 	
    nvmedia_iep_nvscisync.h
 	
NvMediaImage based API:
    NvMediaStatus
    NvMediaIEPNvSciSyncGetVersion(
        NvMediaVersion *version
    );
	This API will be deprecated in DRIVE OS 6.0.4.0 release
	
	The contents of nvmedia_iep_nvscisync.h header file will be merged with nvmedia_iep.h and the resulting API set will have a single version

Header File:
 	
    nvmedia_common_encode.h
    nvmedia_common_encode_decode.h
    nvmedia_common_encode_ofst.h
	The data structures fields will be reorganized and cleaned up, removing unsupported fields previously retained for backward compatibility and making a cohesive grouping of features to enhance ease of use

Table 17.   NvMedia IOFST (DRIVE OS 5.2) to NvMedia IOFA

Module: NvMedia IOFST	
Header File and API	**Comments**

Header File:
 	
    nvmedia_iofst.h
    nvmedia_iofst_nvscisync.h
 	
API:
 	
    NvMediaIOFST*
	NvMedia IOFST APIs use the NVENC HW engine on Xavier DRIVE OS 5.2 to perform Optical Flow and Stereo Disparity Processing Tasks. In Orin DRIVE OS 6.0, the NVENC HW engine no longer supports Optical Flow and Stereo Disparity Processing, and these APIs will be deprecated in DRIVE OS 6.0

	With Orin we introduce a new engine OFA for Optical Flow and Stereo Disparity Processing. The NvMedia IOFA APIs support computation of Optical Flow and Stereo Disparity using the OFA engine. Users of NvMedia IOFST APIs should transition to the NvMedia IOFA APIs newly defined in DRIVE OS 6.0. The look, feel and, usage of the new NvMedia IOFA APIs are similar to prior NvMedia IOFA APIs

Header File:
 	
    nvmedia_iofst.h
 	
API:
 	
    NvMediaStatus
    NvMediaIOFSTGetVersion(
        NvMediaVersion *version
    );
	Header File:
	
	    nvmedia_ofa.h
	
	API:
	
	    NvMediaStatus
	    NvMediaIOFAGetVersion (
	        NvMediaVersion *version
	    );

Module: NvMedia IOFST

Header File and API	Comments

Header File:
 	
   nvmedia_iofst.h
 	
API:
 	
   NvMediaIOFST *
   NvMediaIOFSTCreate(
      const NvMediaDevice *device,
      NvMediaIOFSTType
  estimationType,
      NvMediaSurfaceType inputFormat,
      NvMediaSurfaceType
  outputFormat,
      const
  NvMediaOFSTInitializeParams
  *initParams,
      uint8_t maxInputBuffering,
      NvMediaEncoderInstanceId
  instanceId
   );
	Header File:
	
	   nvmedia_ofa.h
	
	API:
	
	   NvMediaOFA *
	   NvMediaIOFACreate (
	      void
	   );
	Followed by:
	
	   NvMediaStatus
	   NvMediaIOFAInit (
	      const NvMediaDevice
	  *device,
	      NvMediaOfa
	  *ofaPubl,
	      const NvMediaOfaInitParams
	  *initParams,
	      const uint8_t
	  maxInputBuffering
	   );
	
	Some of the existing arguments, such as estimationType, inputFormat, and outputFormat, are members of the NvMediaOfaInitParams structure, along with additional new parameters

Header File:
 	
   nvmedia_iofst.h
 	
API:
 	
   void NvMediaIOFSTDestroy(const NvMediaIOFST *ofst);
	Header File:
	
	   nvmedia_ofa.h
	
	API:
	
	   NvMediaStatus
	   NvMediaIOFADestroy (
	      const NvMediaOfa *ofaPubl
	   );

Header File:
 	
   nvmedia_iofst.h
 	
API:
 	
   NvMediaStatus
   NvMediaIOFSTProcessFrame(
	Header File:
	
	   nvmedia_ofa.h
	
	API:
	
	   NvMediaStatus
	   NvMediaIOFAProcessFrame (

Module: NvMedia IOFST	
Header File and API	**Comments**
| ```
const NvMediaIOFST *ofst,
const NvMediaImage *frame,
const NvMediaImage *refFrame,
const NvMediaImage *mvs,
const
NvMediaOFSTExternalHintParams
*extHintParams,
    NvMediaEncoderInstanceId
instanceId
);
``` | ```
    const NvMediaOfa
*ofaPubl,
    const NvMediaOfaImageArray
*pSurfArray,
    const NvMediaOfaProcessParams
*processParams,
    const NvMediaOfaROIParams
*pROIParams);
```<br><br>Some of the existing arguments, like `frame`, `refFrame`, `mvs`, or analogs of these parameters, are members of the `NvMediaOfaImageArray` structure along with additional new parameters. The `frame`, `reframe` and `mvs` are the image pyramids in case of OFA. Please refer to API documentation for more information. |
| **Header File:**<br><br>    nvmedia_iofst.h<br><br>**API:**<br><br>```
NvMediaStatus
NvMediaIOFSTImageRegister(
    const NvMediaIOFST *ofst,
    const NvMediaImage *image,
    NvMediaAccessMode  accessMode
);
``` | **Header File:**<br><br>nvmedia_ofa.h<br><br>**API:**<br><br>```
NvMediaIOFAImageRegister (
    const NvMediaOfa       *ofaPubl,
    const NvMediaImage     *image,
    const NvMediaAccessMode accessMode
);
``` |
| **Header File:**<br><br>    nvmedia_iofst.h<br><br>**API:**<br><br>```
NvMediaStatus
NvMediaIOFSTImageUnRegister(
    const NvMediaIOFST *ofst,
    const NvMediaImage *image
);
``` | **Header File:**<br><br>    nvmedia_ofa.h<br><br>**API:**<br><br>```
NvMediaIOFAImageUnRegister (
    const NvMediaOfa   *ofaPubl,
    const NvMediaImage *image
);
``` |
| **Header File:**

    nvmedia_iofst.h

API:

N/A | **Header File:**

    nvmedia_ofa.h

API:

(Get SGM configuration in use.) |

Module: NvMedia IOFST	
Header File and API	**Comments**
| | ```
NvMediaStatus
NvMediaIOFAGetSGMConfigParams (
    const NvMediaOfa    *ofaPubl,
    NvMediaOfaSGMParams *pSGMParams
);
```<br><br>(Set SGM configuration)<br><br>```
NvMediaStatus
NvMediaIOFASetSGMConfigParams (
    const NvMediaOfa    *ofaPubl,
    NvMediaOfaSGMParams *pSGMParams
);
```<br><br>Usage: OPTIONAL |
| **Header File:**<br><br>   nvmedia_iofst.h<br><br>**API:**<br>N/A | **Header File:**<br><br>   nvmedia_ofa.h<br><br>**API:**<br><br>(Get OFA Profile Data)<br><br>```
NvMediaIOFAGetProfileData (
    const NvMediaOfa     *ofaPubl,
    NvMediaOfaProfileData *pProfData
);
```<br><br>Usage: OPTIONAL |
| **Header File:**<br><br>   nvmedia_iofst.h<br><br>**API:**<br>N/A | **Header File:**<br><br>   nvmedia_ofa.h<br><br>**API:**<br><br>(Get OFA Capability)<br><br>```
NvMediaStatus
NvMediaIOFAGetCapability (
    const NvMediaOfa     *ofaPubl,
    const NvMediaOfaMode mode,
    NvMediaOfaCapability *pCapability
);
```<br><br>Usage: OPTIONAL |

Table 18.   NvMedia IOFA - NvMediaImage based to NvSciBuf based

Module: NvMedia IOFA	
Header File and API	**Comments**

Header File:
 	
 nvmedia_ofa.h
 	
 NvMediaImage based API:
 	
 NvMediaStatus
 NvMediaIOFAInit (
     const NvMediaDevice
 *device,
     NvMediaOfa
 *ofaPubl,
     const NvMediaOfaInitParams
 *initParams,
     const uint8_t
 maxInputBuffering
 );
	Header File:
	
	 nvmedia_iofa.h
	
	 > NvMediaDevice is deprecated and the argument is removed from the API
	
	 NvSciBuf based API:
	
	 NvMediaStatus
	 NvMediaIOFAInit (
	     NvMediaIofa
	 *ofaPubl,
	     const NvMediaIofaInitParams
	 *initParams,
	     const uint8_t
	 maxInputBuffering
	);

Header File:
 	
 nvmedia_ofa.h
 	
 NvMediaImage based API:
 	
 NvMediaStatus
 NvMediaIOFAProcessFrame (
     const NvMediaOfa
 *ofaPubl,
     const NvMediaOfaImageArray
 *pSurfArray,
     const NvMediaOfaProcessParams
 *processParams,
     const NvMediaOfaROIParams
 *pROIParams
 );
	Header File:
	
	 nvmedia_iofa.h
	
	 > NvMediaOfaImageArray is replaced by NvMediaIofaBufArray which uses NvSciBufObj instead of NvMediaImage.
	
	 NvSciBuf based API:
	
	 NvMediaStatus
	 NvMediaIOFAProcessFrame (
	     const NvMediaIofa
	 *ofaPubl,
	     const NvMediaIofaBufArray
	 *pSurfArray,
	     const NvMediaIofaProcessParams
	 *processParams,
	     const NvMediaIofaROIParams
	 *pROIParams
	);

Header File: nvmedia_ofa.h **NvMediaImage based API:** NvMediaIOFAImageRegister (	**Header File:** nvmedia_iofa.h **NvSciBuf based API:** NvMediaStatus NvMediaIOFARegisterNvSciBufObj (

Module: NvMedia IOFA

Header File and API	Comments

``
      const NvMediaOfa
  *ofaPubl,
      const NvMediaImage   *image,
      const NvMediaAccessMode
  accessMode
  );
``
	``
	      const NvMediaIofa *ofaPubl,
	      NvSciBufObj bufObj
	  );
	``
	NvSciBufObj `replaces` NvMediaImage
	NvMediaAccessMode will be removed and is passed as NvSciBufObj attribute list.

Header File:
    nvmedia_ofa.h
NvMediaImage based API:
``
NvMediaIOFAImageUnRegister (
    const NvMediaOfa   *ofaPubl,
    const NvMediaImage *image
);
``
	Header File:
	    nvmedia_iofa.h
	NvSciBuf based API:
	``
	NvMediaIOFAUnregisterNvSciBufObj (
	    const NvMediaIofa *ofaPubl,
	    NvSciBufObj bufObj
	);
	``
	> NvSciBufObj `replaces` NvMediaImage

Header File:
    nvmedia_iofst_nvscisync.h
 	
API:
 	
NvMediaIOFSTFillNvSciSyncAttrList
NvMediaIOFSTRegisterNvSciSyncObj
NvMediaIOFSTUnregisterNvSciSyncObj
NvMediaIOFSTSetNvSciSyncObjforEOF
NvMediaIOFSTInsertPreNvSciSyncFence
NvMediaIOFSTGetEOFNvSciSyncFence
NvMediaIOFSTSetNvSciSyncObjforSOF
NvMediaIOFSTGetSOFNvSciSyncFence
	In DRIVE OS 5.2 the NvSciSync APIs for IOFST were optional. The newly defined analogues of these APIs for IOFA are mandatory in 6.0 if IOFA needs to be used in a pipeline with other engine APIs.
	Header File:
	    nvmedia_iofa.h
	API:
	
	    NvMediaIOFAFillNvSciSyncAttrList
	    NvMediaIOFARegisterNvSciSyncObj
	    NvMediaIOFAUnregisterNvSciSyncObj
	    NvMediaIOFASetNvSciSyncObjforEOF
	    NvMediaIOFAInsertPreNvSciSyncFence
	    NvMediaIOFAGetEOFNvSciSyncFence
	    NvMediaIOFASetNvSciSyncObjforSOF
	    NvMediaIOFAGetSOFNvSciSyncFence

Header File:
    nvmedia_iofst_nvscisync.h
API:
N/A
	NvMediaIOFAFillNvSciBufAttrList is a new API in DRIVE OS 6.0 to fill IOFA specific attributes in the NvSciBufAttrList. The NvSciBufAttrList is used during creation of NvSciBufObj. This API is mandatory
	
	``
	NvMediaStatus
	NvMediaIOFAFillNvSciBufAttrList(
	    NvSciBufAttrList attrlist
	);
	``

Table 19.　　　NvMedia IJPEG Decode

Module: NvMedia IJPEG Decode	
Header File and API	**Comments**

Header File:
 	
`nvmedia_ijpd.h`
 	
NvMediaImage based API:
`    NvMediaIJPD *`
`    NvMediaIJPDCreate(`
`        const NvMediaDevice *device,`
`        uint16_t maxWidth,`
`        uint16_t maxHeight,`
`        uint32_t maxBitstreamBytes,`
`        uint8_t supportPartialAccel,`
`        NvMediaJPEGInstanceId`
`    instanceId`
`    );`
	> **NvMediaDevice** is deprecated and the argument is removed from the API
	> **NvMediaJPEGInstanceId** parameter is added with Orin. DRIVE OS 6.0 supports two instances of NVJPEG hardware
	
	NvSciBuf based API:
	
	`    NvMediaIJPD *`
	`    NvMediaIJPDCreate(`
	`        uint16_t maxWidth,`
	`        uint16_t maxHeight,`
	`        uint32_t maxBitstreamBytes,`
	`        bool supportPartialAccel,`
	`        NvMediaJPEGInstanceId instanceId`
	`    );`

Header File:
 	
`    nvmedia_ijpd.h`
 	
NvMediaImage based API:
`    NvMediaStatus`
`    NvMediaIJPDRender(`
`        const NvMediaIJPD *decoder,`
`        NvMediaImage *output,`
`        const NvMediaRect *srcRect,`
`        const NvMediaRect *dstRect,`
`        uint8_t downscaleLog2,`
`        uint32_t numBitstreamBuffers,`
`        const NvMediaBitstreamBuffer`
`    *bitstreams,`
`        uint32_t flags,`
`        NvMediaJPEGInstanceId instanceId`
`    );`
	> **NvMediaImage** is deprecated and replaced by **NvSciBufObj**
	> **NvMediaJPEGInstanceId** parameter is added with Orin. DRIVE OS 6.0 supports two instances of NVJPEG hardware
	
	NvSciBuf based API:
	
	`    NvMediaStatus`
	`    NvMediaIJPDRender(`
	`        const NvMediaIJPD *decoder,`
	`        NvSciBufObj target,`
	`        const NvMediaRect *srcRect,`
	`        const NvMediaRect *dstRect,`
	`        uint8_t downscaleLog2,`
	`        uint32_t numBitstreamBuffers,`
	`        const NvMediaBitstreamBuffer`
	`    *bitstreams,`
	`        uint32_t flags,`
	`        NvMediaJPEGInstanceId instanceId`
	`    );`

Header File: `    nvmedia_ijpd.h` **NvMediaImage based API:**	> **NvMediaImage** is deprecated and replaced by **NvSciBufObj**. > **NvMediaJPEGInstanceId** parameter is added with Orin. DRIVE OS 6.0 supports two instances of NVJPEG hardware.

Module: NvMedia IJPEG Decode	
Header File and API	**Comments**
| ```
NvMediaStatus
NvMediaIJPDRenderYUV(
    const NvMediaIJPD *decoder,
    NvMediaImage *output,
    uint8_t downscaleLog2,
    uint32_t numBitstreamBuffers,
    const NvMediaBitstreamBuffer
*bitstreams,
    uint32_t flags,
    NvMediaJPEGInstanceId instanceId
);
``` | **NvSciBuf based API:**<br><br>```
NvMediaStatus
NvMediaIJPDRenderYUV(
    const NvMediaIJPD *decoder,
    NvSciBufObj target,
    uint8_t downscaleLog2,
    uint32_t numBitstreamBuffers,
    const NvMediaBitstreamBuffer
*bitstreams,
    uint32_t flags,
    NvMediaJPEGInstanceId instanceId
);
``` |
| **Header File:**

    nvmedia_ijpd_nvscisync.h

API:

    NvMediaIJPDFillNvSciBufAttrs
    NvMediaIJPDNvSciSyncGetVersion
    NvMediaIJPDFillNvSciSyncAttrList
    NvMediaIJPDRegisterNvSciSyncObj
    NvMediaIJPDUnregisterNvSciSyncObj
    NvMediaIJPDSetNvSciSyncObjforEOF
    NvMediaIJPDInsertPreNvSciSyncFence
    NvMediaIJPDGetEOFNvSciSyncFence
    NvMediaIJPDSetNvSciSyncObjforSOF
    NvMediaIJPDGetSOFNvSciSyncFence | DRIVE OS 6.0 introduces new APIs for synchronization that are analogous to 5.2 NvMedia IEP synchronization APIs. The 6.0 APIs are mandatory when the NvMedia JPEG Decoder is used in a pipeline scenario |

Table 20.   NvMedia IJPEG Encode

Module: NvMedia IJPEG Encode	
Header File and API	**Comments**

Header File:
 	
    nvmedia_ijpe.h
 	
API:
 	
    NvMediaIJPE *
    NvMediaIJPECreate(
        const NvMediaDevice *device,
        NvMediaSurfaceType inputFormat,
	> NvMediaDevice is deprecated in 6.0 and the argument is removed from the API
	
	> NvMediaJPEGInstanceId parameter is added with Orin. DRIVE OS 6.0 supports two instances of NVJPEG hardware
	
	> An analogous type replaces the NvMediaSurfaceType parameter

Module: NvMedia IJPEG Encode	
Header File and API	**Comments**

`uint8_t maxOutputBuffering,`
`uint32_t  maxBitstreamBytes,`
`NvMediaJPEGInstanceId`
`instanceId`
`  );`	

Header File:
 	
`nvmedia_ijpe.h`
 	
API:
 	
`NvMediaStatus`
`NvMediaIJPEFeedFrameQuant(`
`    const NvMediaIJPE *encoder,`
`    NvMediaImage *frame,`
`    uint8_t *lumaQuant,`
`    uint8_t *chromaQuant,`
`    NvMediaJPEGInstanceId`
`instanceId`
`);`
	> NvMediaImage is deprecated and replaced by NvSciBufObj
	
	> NvMediaJPEGInstanceId parameter is added with **Orin**. Drive OS 6.0 supports two instances of NVJPEG hardware

Header File:
 	
`nvmedia_ijpe.h`
 	
API:
 	
`NvMediaStatus`
`NvMediaIJPEFeedFrame(`
`    const NvMediaIJPE *encoder,`
`    NvMediaImage *frame,`
`    uint8_t quality,`
`    NvMediaJPEGInstanceId`
`instanceId`
`);`
	> NvMediaImage is deprecated and replaced by NvSciBufObj
	
	> NvMediaJPEGInstanceId parameter is added with Orin. DRIVE OS 6.0 supports two instances of NVJPEG hardware

`Header File:`
`nvmedia_ijpe.h`
 	
`API:`
`NvMediaStatus`
`NvMediaIJPEFeedFrameRateControl(`
`    const NvMediaIJPE *encoder,`
`    NvMediaImage *frame,`
`    uint8_t *lumaQuant,`
`    uint8_t *chromaQuant,`
`    uint32_t targetImageSize,`
`    NvMediaJPEGInstanceId`
`instanceId`
	> NvMediaImage is deprecated and replaced by NvSciBufObj
	
	> NvMediaJPEGInstanceId parameter is added with Orin. DRIVE OS 6.0 supports two instances of NVJPEG hardware

Module: NvMedia IJPEG Encode	
Header File and API	**Comments**
`       );`	

Header File:
 	
`    nvmedia_ijpe_nvscisync.h`
 	
API:
 	
`    NvMediaIJPEFillNvSciBufAttrs`
`    NvMediaIJPENvSciSyncGetVersion`
`    NvMediaIJPEFillNvSciSyncAttrList`
`    NvMediaIJPERegisterNvSciSyncObj`
`    NvMediaIJPEUnregisterNvSciSyncObj`
`    NvMediaIJPESetNvSciSyncObjforEOF`
`    NvMediaIJPEInsertPreNvSciSyncFence`
`    NvMediaIJPEGetEOFNvSciSyncFence`
`    NvMediaIJPESetNvSciSyncObjforSOF`
`    NvMediaIJPEGetSOFNvSciSyncFence`
	DRIVE OS 6.0 introduces new APIs for synchronization that are analogous to 5.2 NvMedia IEP synchronization APIs. The 6.0 APIs are mandatory when the NvMedia JPEG Encoder is used in a pipeline scenario

Table 21. NvMedia VPI

Module: NvMedia VPI	
Header File and API	**Comments**

Header File:
 	
`    nvmedia_vpi.h`
`    nvmedia_vpi_nvscisync.h`
 	
API:
 	
`    NvMediaVPI*`
	In NVIDIA DRIVE OS 5.2, NvMedia VPIs use PVA engines to perform fixed computer vision functions.In 6.0, NvMedia VPIs are removed. To use the PVA engine, use DriveWorks or cuPVA

Module: NvMedia Image Pyramid	
Header File and API	**Comments**

Header File:
 	
`    nvmedia_image_pyramid.h`
`    nvmedia_image_pyramid_nvscisync.h`
 	
API:
 	
`    NvMediaVPI*`
	NvMedia Image Pyramid, which is only used by NvMedia VPI, is removed in DRIVE OS 6.0

Module: NvMedia Array	
Header File and API	**Comments**

Module: NvMedia VPI	
Header File and API	**Comments**

Header File:
 	
    nvmedia_array.h
    nvmedia_array_nvscibuf.h
 	
API:
 	
    NvMediaArray*
	NvMedia Array, which is only used by NvMedia VPI, is removed in DRIVE OS 6.0.

Header File:
 	
    nvmedia_array.h
    nvmedia_arraymetadata.h
    nvmedia_array_nvscibuf.h
 	
API:
 	
    NvMediaArray*
	NvMedia Array, which is only used by NvMedia VPI, is removed in DRIVE OS 6.0.

Table 22.   Camera/SIPL API Changes

Several APIs and structures were updated.

File	Current API or Structure that will Change	Change Description and Notes
NvSIPLCamera.hpp	virtual SIPLStatus INvSIPLCamera::GetImageAttributes(uint32_t index, INvSIPLClient::ConsumerDesc::OutputType const outType, **NvSIPLImageAttr &imageAttr**) = 0;	virtual SIPLStatus nvsipl::INvSIPLCamera::GetImageAttributes(uint32_t index, INvSIPLClient::ConsumerDesc::OutputType const outType, **NvSciBufAttrList & imageAttr**)
NvSIPLCamera.hpp	virtual SIPLStatus INvSIPLCamera::RegisterImageGroups(uint32_t index, const std::vector<**NvMediaImageGroup*>** &imageGroups) = 0;	virtual SIPLStatus nvsipl::INvSIPLCamera::RegisterImages(uint32_t const index, INvSIPLClient::ConsumerDesc::OutputType const outType,

File	Current API or Structure that will Change	Change Description and Notes
		const std::vector< **NvSciBufObj** > & images)
NvSIPLCamera.hpp	virtual SIPLStatus INvSIPLCamera::RegisterImages(uint32_t index, INvSIPLClient::ConsumerDesc::OutputType const outType, const std::vector<**NvMediaImage***> &images) = 0;	virtual SIPLStatus nvsipl::INvSIPLCamera::RegisterImages(uint32_t const index, INvSIPLClient::ConsumerDesc::OutputType const outType, const std::vector< **NvSciBufObj** > & images
NvSIPLCamera.hpp	virtual SIPLStatus INvSIPLCamera::FillNvSciSyncAttrList(uint32_t index, INvSIPLClient::ConsumerDesc::OutputType const outType, NvSciSyncAttrList const attrList, **NvMediaNvSciSyncClientType** const clientType) = 0;	virtual SIPLStatus nvsipl::INvSIPLCamera::FillNvSciSyncAttrList(uint32_t index, INvSIPLClient::ConsumerDesc::OutputType const outType, NvSciSyncAttrList const attrList, **NvSiplNvSciSyncClientType** const clientType)
NvSIPLCamera.hpp	NvSiplNvSciSyncClientType::SIPL_SIGNALER_WAITER NvSiplNvSciSyncObjType::NVSIPL_EOF_PRESYNCOBJ NvSiplNvSciSyncObjType::NVSIPL_SOF_PRESYNCOBJ NvSiplNvSciSyncObjType::NVSIPL_SOFSYNCOBJ	These symbols are deprecated
NvSIPLCamera.hpp	virtual SIPLStatus INvSIPLCamera::RegisterNvSciSyncObj(uint32_t index, INvSIPLClient::ConsumerDesc::OutputType const outType, **NvMediaNvSciSyncObjType** const syncobjtype, NvSciSyncObj const syncobj) = 0;	virtual SIPLStatus nvsipl::INvSIPLCamera::RegisterNvSciSyncObj(uint32_t index, INvSIPLClient::ConsumerDesc::OutputType const outType, **NvSiplNvSciSyncObjType** const syncobjtype, NvSciSyncObj const syncobj

File	Current API or Structure that will Change	Change Description and Notes
		)
`NvSIPLPipelineMgr.hpp`	`NvSIPLImageGroupWriter::RawBuffer::`**`NvMediaImageGroup`** `*imageGroup;`	`NvSciBufObj image;` `uint32_t uIndex;` `bool discontinuity;` `bool dropBuffer;` `uint64_t frameCaptureTSC;` `uint64_t frameCaptureStartTSC;` `};`
`NvSIPLPipelineMgr.hpp`	**`NvSIPLImageAttr`**	The structure will be removed. Stop using it and migrate to the newly provided `NvSciBuf` attribute

`NvSIPLPipelineMgr.hpp`	`NvSIPLDownscaleCropCfg::`**`NvMediaRect`** `ispInputCrop;`
	
	 `NvSIPLDownscaleCropCfg::`**`NvMediaRect`** `isp0OutputCrop;`
	
	 `NvSIPLDownscaleCropCfg::`**`NvMediaRect`** `isp1OutputCrop;`
		`NvSIPLDownscaleCropCfg::`**`NvSIPLRect`** `ispInputCrop;`
		
		 `NvSIPLDownscaleCropCfg::`**`NvSIPLRec`**`t isp0OutputCrop;`
		
		 `NvSIPLDownscaleCropCfg::`**`NvSIPLRect`** `isp1OutputCrop;`

`NvSIPLClient.hpp`	`INvSIPLClient::INvSIPLClient::ImageMetaData::`**`NvMediaISPBadPixelStatsData`** `badPixelStats`
	
	 `INvSIPLClient::INvSIPLClient::ImageMetaData::`**`NvMediaISPHistogramStatsData`** `histogramStats[2];`
	
	 `INvSIPLClient::INvSIPLClient::ImageMetaData::`**`NvMediaISPHistogramStats`** `histogramSettings[2];`
	
	 `INvSIPLClient::INvSIPLClient::ImageMetaData::`**`NvMediaISPLocalAvgClipStatsData`** `localAvgClipStats[2];`
	
	 `INvSIPLClient::INvSIPLClient::ImageMetaData::`**`NvMediaISPLocalAvgClipStats`** `localAvgClipSettings[2];`
		`INvSIPLClient::INvSIPLClient::ImageMetaData::` **`NvSiplISPBadPixelStatsData`** `badPixelStats`
		
		 `INvSIPLClient::INvSIPLClient::ImageMetaData::` **`NvSiplISPHistogramStatsData`** `histogramStats[2];`
		
		 `INvSIPLClient::INvSIPLClient::ImageMetaData::` **`NvSiplISPHistogramStats`** `histogramSettings[2];`
		
		 `INvSIPLClient::INvSIPLClient::ImageMetaData::` **`NvSiplISPLocalAvgClipStatsData`** `localAvgClipStats[2];`
		
		 `INvSIPLClient::INvSIPLClient::ImageMetaData::`**`NvMediaISPL`**

File	Current API or Structure that will Change	Change Description and Notes

	`INvSIPLClient::INvSIPLClient::Im ageMetaData::NvSiplGlobalTime captureGlobalTimeStamp;`
	
	`INvSIPLClient::INvSIPLClient::Im ageMetaData:: NvSiplTimeBase timeBase;`
		ocalAvgClipStats `localAvgClipSettings[2];`
		
		Removed in 6.0.6.0
		
		Removed in 6.0.6.0

`NvSiplControlAutoDef .hpp`	`SiplControlIspStatsSetting::`**NvMe diaISPLocalAvgClipStats** `lac[2];`
	
	`SiplControlIspStatsSetting::`**NvMe diaISPHistogramStats** `hist1;`
	
	`SiplControlIspStatsSetting::`**NvMe diaISPFlickerBandStats** `fbStats;`
		`SiplControlIspStatsSetting::` **NvSiplISPLocalAvgClipStats** `lac[2];`
		
		`SiplControlIspStatsSetting::` **NvSiplISPHistogramStats** `hist1;`
		
		`SiplControlIspStatsSetting::` **NvSiplISPFlickerBandStats** `fbStats;`

`NvSIPLDeviceBlockInf o.hpp`	`SensorInfo::VirtualChannelInfo::` **NvMediaICPInputFormatType** `inputFormat`
	
	`DeviceBlockInfo::isSimulatorMode Enabled`
		`SensorInfo::VirtualChannelIn fo::` **NvSiplCapInputFormatType** `inputFormat.`
		
		`DeviceBlockInfo::CameraModul eInfo::isSimulatorModeEnable d`

`CNvMDeviceBlockInfo. hpp`	`DeviceBlockInfo::`**NvMediaICPInter faceType** `csiPort`
	
	`DeviceBlockInfo::`**NvMediaICPCsiPh yMode** `phyMode`
		`DeviceBlockInfo::` **NvSiplCapInterfaceType** `csiPort`
		
		`DeviceBlockInfo::` **NvSiplCapCsiPhyMode** `phyMode`

`devblk_cdi.h`	`DevBlkCDIPWL::`**NvMediaPointDouble** `kneePoints[DEVBLK_CDI_MAX_PWL_KN EEPOINTS]`	`DevBlkCDIPWL::` **NvSiplPointDouble** `kneePoints[DEVBLK_CDI_MAX_PW L_KNEEPOINTS]`
`CNvMDeserializer.hpp`	`CNvMCameraModuleCommon::SensorCo nnectionProperty::`**NvMediaICPInpu tFormat** `inputFormat`	`CNvMCameraModuleCommon::Sens orConnectionProperty::`**NvSipl CapInputFormat** `inputFormat`

File	Current API or Structure that will Change	Change Description and Notes
	`CNvMCameraModule::CameraModuleConfig::`**`NvMediaICPInterfaceType`**` eInterface;`	`CNvMCameraModule::CameraModuleConfig::`**`NvSiplICapInterfaceType`**` eInterface;`
	`CNvMCameraModule::Property::SensorProperty::`**`NvMediaICPInputFormat`**` inputFormat;`	`CNvMCameraModule::Property::SensorProperty::`**`NvSiplCapInputFormat`**` inputFormat;`
	`CNvMCameraModule::Property::SensorProperty::`**`NvMediaSurfaceType`**` surfaceType;`	`CNvMCameraModule::Property::SensorProperty::`**`NvSiplCapInputFormat`**` inputFormat;`
	`CNvMDeserializer::DeserializerParams::`**`NvMediaICPInterfaceType`**` eInterface;`	`CNvMDeserializer::DeserializerParams::`**`NvSiplCapInterfaceType`**` eInterface;`
	`CNvMDeserializer::DeserializerParams::`**`NvMediaICPCsiPhyMode`**` ePhyMode;`	`CNvMDeserializer::DeserializerParams::`**`NvSiplCapCsiPhyMode`**` ePhyMode;`
	`CNvMDeserializer::`**`NvMediaICPInterfaceType`**` m_eInterface;`	`CNvMDeserializer::`**`NvSiplCapCsiPhyMode`**` m_ePhyMode;`
	`CNvMDeserializer::`**`NvMediaICPCsiPhyMode`**` m_ePhyMode;`	
`nvmedia_icp_structs.h`	Header file changes	Internals will be redefined Stop using the header file; a new header file, NvSIPLCapStructs.h, with updated structures and enumeration will be provided. This header contains structures and enumerations mentioned earlier in this document
`nvmedia_isp_stat.h`	Header file changes	Internals will be redefined Stop using the header file; a new header file, NvSIPLISPStat.hpp, with updated structures and enumeration will be provided. This header contains structures and enumerations mentioned earlier in this document

File	Current API or Structure that will Change	Change Description and Notes

NvMediaTensor APIs that take NvMediaDevice as input parameters were updated.

> **Note:** **The NvMediaDevice struct is typedef to void, and the APIs, NvMediaDeviceCreate() and NvMediaDeviceDestroy(), are marked deprecated and will be removed in 6.0.5.0 along with the libraries that contain them -- libnvmedia.so and libnvmedia_core.so.**
>
> **The updated versions of the following NvMediaTensor APIs will require a NULL value to be passed as input, instead of a valid NvMediaDevice pointer.**
>
> NvMediaTensorCreateFromNvSciBuf() and
>
> NvMediaTensorFillNvSciBufAttrs()
>
> **A recommendation for NvMediaTensor clients is to remove the use of NvMediaDeviceCreate() and NvMediaDeviceDestroy() APIs in their applications. Clients should also stop using libnvmedia.so and libnvmedia_core.so in their liner/makefiles.**
>
> **The API signature has not changed for any of these NvMediaTensor APIs, but the implementation has changed. You must use the recompiled library.**
>
> **The nvm_dlaSample application, which is part of the PDK/SDK, is updated to remove NvMediaDevice APIs and NULL value is passed as input for the above NvMedia Tensor APIs, where a NvMediaDevice handle is expected.**

Table 23.   NvMediaTensor API Changes

Module: NvMedia Tensor	
Header File and API	**Comments**
Header File: nvmedia_tensor_nvscibuf.h **API:** NvMediaStatus NvMediaTensorCreateFromNvSciBuf(NvMediaDevice *device, NvSciBufObj nvSciBufObjInstance, NvMediaTensor **nvmTensor);	> NvMediaDevice is deprecated in 6.0 and the dependency has been removed. > A NULL value is passed as input instead of a valid NvMediaDevice pointer. **Example:** NvMediaStatus status; NvSciBufObj bufObj; NvMediaTensor *tensor; status = NvMediaTensorCreateFromNvSciBuf(NULL,bufobj, &tensor);
Header File:	> NvMediaDevice is being deprecated in 6.0 and the dependency on it has been removed.

Module: NvMedia Tensor	
Header File and API	**Comments**

`nvmedia_tensor_nvscibuf.h`
 	
API:
 	
`NvMediaStatus`
`NvMediaTensorFillNvSciBufAttrs(`
    `const NvMediaDevice *device,`
    `const NvMediaTensorAttr`
`*attrs,`
    `uint32_t numAttrs,`
    `uint32_t flags,`
    `NvSciBufAttrList attr_h`
`);`
	> A NULL value is to be passed as input instead of a valid NvMediaDevice pointer.
	
	Example:
	
	`NvMediaStatus status;`
	`NVM_TENSOR_DEFINE_ATTR(tensorAttr);`
	`uint32_t numTensorAttr;`
	`NvSciBufAttrList attrlist;`
	`status = NvMediaTensorFillNvSciBufAttrs(`
	`NULL,tensorAttr,numTensorAttr,0,`
	`attrlist);`

NvMediaImage to NvSciBuf Migration

In NVIDIA DRIVE 6.0, `NvMediaImage` type is deprecated and replaced by `NvSciBuf`. All client applications using `NvMediaImage` structure must migrate to `NvSciBuf`. NvMedia Engine APIs will also use `NvSciBuf`.

In NVIDIA DRIVE OS 5.2, explicit engine synchronization using `NvSciSync` objects was optional but in NVIDIA DRIVE 6.0, it is mandatory.

Table 24.   Deprecated NvMediaImage APIs

Deprecated NvMediaImage-Related APIs
`NvMediaSurfaceGetVersion`
`NvMediaSurfaceFormatGetType`
`NvMediaSurfaceFormatGetAttrs`
`NvMediaImageGetVersion`
`NvMediaImageCreateNew`
`NvMediaImageDestroy`
`NvMediaImageGetEmbeddedData`
`NvMediaImageGetStatus`
`NvMediaImageSetTag`
`NvMediaImageGetTag`
`NvMediaImageGetTimeStamp`

Deprecated NvMediaImage-Related APIs
NvMediaImageGetGlobalTimeStamp
NvMediaImageGetTimeBase
NvMediaImageLock
NvMediaImageUnlock
NvMediaImagePutBits
NvMediaImageGetBits

Table 25. Migration from NvMediaImage to NvSciBuf

`NvSciBuf` has a similar model of allocating buffers. It is based on attributes, and the following table gives an overview of how to migrate `NvMediaImage` attributes to `NvSciBuf`.

NvMediaImage Attributes	NvSciBuf Attributes
NVM_SURF_ATTR_WIDTH	NvSciBufImageAttrKey_PlaneWidth
NVM_SURF_ATTR_HEIGHT	NvSciBufImageAttrKey_PlaneHeight
NVM_SURF_ATTR_MIN_EMB_LINES_TOP	NvSciBufImageAttrKey_TopPadding
NVM_SURF_ATTR_MIN_EMB_LINES_BOTTOM	NvSciBufImageAttrKey_BottomPadding
NVM_SURF_ATTR_CPU_ACCESS	NvSciBufGeneralAttrKey_NeedCpuAccess NvSciBufGeneralAttrKey_EnableCpuCache NvSciBufGeneralAttrKey_CpuNeedSwCacheCoherency
NVM_SURF_ATTR_ALLOC_TYPE	N/A
NVM_SURF_ATTR_PEER_VM_ID	N/A
NVM_SURF_ATTR_SCAN_TYPE	NvSciBufImageAttrKey_ScanType
NVM_SURF_ATTR_COLOR_STD_TYPE	NvSciBufImageAttrKey_PlaneColorStd
NVM_SURF_ATTR_SURF_TYPE	NvSciBufImageAttrKey_PlaneColorFormat
NVM_SURF_ATTR_LAYOUT	NvSciBufImageAttrKey_Layout
NVM_SURF_ATTR_DATA_TYPE	NvSciBufImageAttrKey_PlaneDatatype
NVM_SURF_ATTR_MEMORY	NvSciBufImageAttrKey_PlaneCount

NvMediaImage Attributes	NvSciBuf Attributes
NVM_SURF_ATTR_SUB_SAMPLING_TYPE	NvSciBufImageAttrKey_PlaneCount NvSciBufImageAttrKey_PlaneColorFormat NvSciBufImageAttrKey_PlaneBitsPerPixel
NVM_SURF_ATTR_BITS_PER_COMPONENT	NvSciBufImageAttrKey_PlaneBitsPerPixel
NVM_SURF_ATTR_COMPONENT_ORDER	NvSciBufImageAttrKey_PlaneColorFormat

> **Note:** Not all NvMediaImage attributes map directly to NvSciBuf attributes. Refer to the NvSciBuf documentation.

NvMedia EGL Stream to NvSciStreams Migration

In NVIDIA DRIVE OS 5.2, NvMedia APIs natively support sending and receiving `NvMediaVideoSurface` and `NvMediaImage` type surfaces over an EGL Stream connection both as a producer and a consumer.

In NVIDIA DRIVE 6.0, `NvMediaVideoSurface` and `NvMediaImage` types, and NvMedia EGL Stream support are deprecated. `NvMediaVideoSurface` and `NvMediaImage` types are replaced by `NvSciBuf`, EGL Stream support is replaced by NvSciStreams.

Table 26.   Deprecated NvMedia EGL Stream APIs in NVIDIA DRIVE 6.0

Deprecated NvMedia EGL Stream APIs
NvMediaEglStreamGetVersion
NvMediaEglStreamProducerCreate
NvMediaEglStreamProducerSetAttributes
NvMediaEglStreamProducerDestroy
NvMediaEglStreamProducerPostSurface
NvMediaEglStreamProducerGetSurface
NvMediaEglStreamConsumerCreate
NvMediaEglStreamConsumerDestroy
NvMediaEglStreamConsumerAcquireSurface
NvMediaEglStreamConsumerReleaseSurface
NvMediaEglStreamProducerPostImage

Deprecated NvMedia EGL Stream APIs
NvMediaEglStreamProducerGetImage
NvMediaEglStreamConsumerAcquireImage
NvMediaEglStreamConsumerReleaseImage
NvMediaEglStreamProducerPostMetaData
NvMediaEglStreamConsumerAcquireMetaData

EGL Stream and NvSciStreams are both based on a producer/consumer model. When a stream connection is established between a producer and a consumer, the producer posts buffer objects to the stream, which is received by the consumer. Refer to the NvSciStreams documentation regarding establishing stream communication between a producer and consumer.

The following table shows a typical high-level NvMedia Producer programming sequence and the corresponding NvSciStreams equivalent. This sequence does not provide details regarding the transfer of synchronization objects.

Table 27.   NvMedia Producer and NvSciStreams Programming Sequences

NvMedia EGL Stream Producer	NvSciStreams Producer
```// Generate Image … // Post Image to EGL Stream NvMediaEglStreamProducerPostImage() // Get the Image back from EGL Stream NvMediaEglStreamProducerGetImage()```	```// Query event NvSciStreamBlockEventQuery() // Get packet to get index to Image NvSciStreamProducerPacketGet() // Generate Image … // Post Image to NvSciStreams NvSciStreamProducerPacketPresent()```

The following table shows a typical high-level NvMedia Consumer programming sequence and the corresponding NvSciStreams equivalent. This sequence does not provide details regarding the transfer of synchronization objects.

Table 28.   NvMedia Consumer and NvSciStreams Programming Sequences

NvMedia EGL Stream Consumer	NvSciStreams Consumer
```// Get frame status eglQueryStreamKHR() // Receive Image from EGL Stream NvMediaEglStreamConsumerAcquireImage() // Process Image … // Release Image back to EGL Stream```	```// Wait for event NvSciStreamBlockEventQuery() // Receive packet to get index to Image NvSciStreamConsumerPacketAcquire() // Process Image …```

NvMedia EGL Stream Consumer	NvSciStreams Consumer

`NvMediaEglStreamConsumerReleaseImage()`	`// Release Image back to`
	`NvSciStreams`
	`NvSciStreamConsumerPacketRelease()`

NvMedia Image Processing Pipeline (IPP) Changes Introduced in DRIVE OS 5.2

NvMedia Image Processing Pipeline (IPP) APIs are removed. Instead, use NvMedia SIPL APIs. The deprecated APIs and the associated sample applications will be removed in the 5.2 release.

In NVIDIA DRIVE OS 5.1, the NvMedia Image Processing Pipeline (IPP) framework provides the interface to capture RAW images from camera sensors and process them using the NVIDIA DRIVE AGX Xavier™ hardware Image Signal Processing (ISP).

The framework provides various APIs to set up image capture and processing pipelines. NvMedia IPP APIs are typically used in conjunction with ExtImgDev APIs to setup the complete image capture and processing pipelines. ExtImgDev APIs provide interfaces to program the external image devices like image sensors, serializers and de-serializers.

NVIDIA DRIVE™ OS 5.1 platforms have been supporting a new camera framework called SIPL designed for safety use cases. The SIPL framework provides a single set of C++ interfaces to instantiate imaging pipeline(s) in Xavier-based platforms. It supports both raw image capture and ISP processed output.

This section includes a comparison of API sequences for typical image processing use cases.

All unused or unsupported legacy components, such as ISC and VMP, are deprecated.

Capture and Process from a Live Camera

This section details the API call sequence comparison for capturing and processing of frames from a live camera.

Table 29.   NvMedia IPP and SIPL API Call Sequence Comparison

Phase	NvMedia IPP API See nvmipp_raw source for details.	NvMedia SIPL API See nvsipl_camera source for details.

Initialization	1. Set up ExtImgDev params
	
	`ExtImgDevInit`
	`NvMediaIPPManagerCreate`
	`NvMediaIPPPipelineCreate`
		1. Create INvSIPLCamera instance
		
		`INvSIPLCamera::GetInstance`

Phase	NvMedia IPP API See nvmipp_raw source for details.	NvMedia SIPL API See nvsipl_camera source for details.

	2. Create sensor control component.
	
	`NvMediaIPPComponentCreateNew (ISC)`
	
	3. Create capture component using ExtImgDev properties
	
	`NvMediaIPPComponentCreateNew (ICP)`
	`NvMediaIPPComponentAddToPipeline`
	
	4. Specify the buffer pool properties and ISP output properties
	
	`NvMediaIPPComponentCreateNew (ISP`
	
	5. Specify the algorithm configuration
	
	`NvMediaIPPComponentCreateNew (ALG)`
	
	6. Create the output component
	
	`NvMediaIPPComponentCreateNew (OUTPUT)`
	
	7. Use multiple `NvMediaIPPComponentAttach` to attach components
		2. Set up Platform config data structure to describe the camera. INvSIPLQuery interfaces can be used to fetch one of the supported platform configurations
		
		`INvSIPLCamera::SetPlatformCfg`
		
		3. Set pipeline config describing what outputs are needed. Get back queues to receive the output images
		
		`INvSIPLCamera::SetPipelineCfg`
		
		4. Initialize the camera
		
		`INvSIPLCamera::Init`
		
		5. Get the image attributes.
		
		`INvSIPLCamera::GetImageAttributes`
		
		6. Allocate buffers using NvSciBuf APIs
		Register the buffers with SIPL
		
		`INvSIPLCamera::RegisterImages`
		
		7. Get NvSciSync attributes
		
		`INvSIPLCamera::FillNvSciSyncAttrList`
		
		8. Allocate NvSciSyncObjs using NvSciSync APIs. Register NvSciSyncObjs with SIPL
		
		`INvSIPLCamera::RegisterNvSciSyncObj`
		
		9. Register the Auto Control plugin and ISP calibration data to use
		
		`INvSIPLCamera::RegisterAutoControlPlugin`

Phase	NvMedia IPP API See nvmipp_raw source for details.	NvMedia SIPL API See nvsipl_camera source for details.

Start	1. Start the pipeline.
	
	`NvMediaIPPPipelineStart`
	
	2. Start ExtImgDev.
	
	`ExtImgDevStart`
		1. Start the camera.
		
		`INvSIPLCamera::Start`

Runtime	1. In an application thread:
	
	1a. Get the output from NvMedia IPP.
	
	`NvMediaIPPComponentGetOutput`
	
	1b. Consume the output
	1c. Return the consumed buffer back to IPP
	
	`NvMediaIPPComponentReturnOutput`
		1. In an application thread:
		
		1a. Get the output buffer from the queue(s) returned in the SetPipelineCfg API
		
		`INvSIPLFrameCompletionQueue::Get`
		
		1b. Get the post-fence of the buffer
		
		`INvSIPLBuffer::GetEOFNvSciSyncFence`
		
		1c. Wait on the returned on the post-fence and consume the buffer
		
		1d. Add pre-fences to the buffer (if any)
		
		`INvSIPLBuffer::AddNvSciSyncPrefence`
		
		1e. Release the reference.
		
		`INvSIPLBuffer::Release`

Reprocess a RAW File Using Hardware ISP

This section details the API call sequence comparison for reprocessing a RAW file using hardware ISP.

Table 30.   **Reprocessing a RAW File using Hardware ISP**

Phase	NvMedia IPP API See nvmipp_raw source for details.	NvMedia SIPL API See nvsipl_camera source for details.

Initialization	1. Set up ExtImgDev params
	
	`ExtImgDevInit`
	`NvMediaIPPManagerCreate`
	`NvMediaIPPPipelineCreate`
	
	2. Create the sensor control component.
	
	`NvMediaIPPComponentCreateNew (ISC)`
	
	3. Create file reader component.
	
	`NvMediaIPPComponentCreateNew (FILE_READER)`
	
	4. Specify buffer pool properties and ISP output properties.
	
	`NvMediaIPPComponentCreateNew (ISP)`
	
	5. Specify algorithm configuration.
	
	`NvMediaIPPComponentCreateNew (ALG)`
	
	6. Create output component.
	
	`NvMediaIPPComponentCreateNew (OUTPUT)`
	
	7. Use multiple `NvMediaIPPComponentAttach` to attach components
		1. Create INvSIPLCamera instance
		
		`INvSIPLCamera::GetInstance`
		
		2. Set up Platform config data structure to describe the camera. INvSIPLQuery interfaces can be used to fetch one of the supported platform configurations
		
		`INvSIPLCamera::SetPlatformCfg`
		
		3. Set pipeline config describing what outputs are needed and image group writer callback. Get back queues to receive the output images
		
		`INvSIPLCamera::SetPipelineCfg`
		
		4. Initialize the camera.
		
		`INvSIPLCamera::Init`
		
		5. Get the image attributes
		
		`INvSIPLCamera::GetImageAttributes`
		
		6. Allocate buffers using NvSciBuf APIs
		Register the buffers with SIPL
		
		`INvSIPLCamera::RegisterImages`
		
		7. Get NvSciSync attributes.

Phase	NvMedia IPP API See nvmipp_raw source for details.	NvMedia SIPL API See nvsipl_camera source for details.

		`INvSIPLCamera::FillNvSciSyncAttrList`
		
		8. Allocate NvSciSyncObjs using NvSciSync APIs. Register NvSciSyncObjs with SIPL.
		
		`INvSIPLCamera::RegisterNvSciSyncObj`
		
		9. Register the Auto Control plugin and ISP calibration data to use
		
		`INvSIPLCamera::RegisterAutoControlPlugin`

Start	1. Start the pipeline.
	
	`NvMediaIPPPipelineStart`
	2. Start ExtImgDev.
	
	`ExtImgDevStart`
		1. Start the camera.
		
		`INvSIPLCamera::Start`

Runtime	1. In an application thread:
	
	1a. Get the output from NvMedia IPP.
	
	`NvMediaIPPComponentGetOutput`
	
	1b. Consume the output.
	
	1c. Return the consumed buffer back to IPP.
	
	`NvMediaIPPComponentReturnOutput`
		1. Populate the image with RAW data in the image group writer callback implemented by the app and register with the SIPL
		
		2. In an application thread:
		
		2a. Get the output buffer from the queue(s) returned in the SetPipelineCfg API.
		
		`INvSIPLFrameCompletionQueue::Get`
		
		1b. Get the post-fence of the buffer
		
		`INvSIPLBuffer::GetEOFNvSciSyncFence`
		
		1c. Wait on the returned on the post-fence and consume the buffer

Phase	NvMedia IPP API See nvmipp_raw source for details.	NvMedia SIPL API See nvsipl_camera source for details.

		1d. Add pre-fences to the buffer (if any).
		
		`INvSIPLBuffer::`
		`AddNvSciSyncPrefence`
		
		1e. Release the reference.
		
		`INvSIPLBuffer::Release`

NvMedia Array and NvMedia CVScratchPad Deprecation

In NVIDIA DRIVE 6.0, NvMedia Array and NvMedia CVScratchPad APIs are deprecated. These APIs are intended for use with Legacy NvMedia IOFST(Array) and NvMedia VPI(Array and CVScratchPad) components, which are not are not part of NVIDIA DRIVE 6.0..

NvMedia ISC Deprecation

In NVIDIA DRIVE 6.0, NvMedia ISC APIs are deprecated. These APIs were intended for use with Legacy NvMedia IPP component, which is not part of NVIDIA DRIVE 6.0..

NvMedia Core Deprecation

In NVIDIA DRIVE 6.0, `NvMediaDeviceCreate` and `NvMediaDeviceDestroy` APIs are marked as deprecated and removed in 6.0.5.0 along with the libraries that contain them – `libnvmedia.so` and `libnvmedia_core.so`. The `NvMediaDevice` handle did not carry valuable information and was retained in NVIDIA DRIVE 5.2 for legacy reasons.

Clients should remove the use of `NvMediaDeviceCreate` and `NvMediaDeviceDestroy` APIs in applications. Additionally, clients should stop using `libnvmedia.so` and `libnvmedia_core.so` in linker/makefiles.

`NvMediaCoreGetVersion` and `NvMediaReleaseVersion` APIs are deprecated and removed. These APIs don't add value because the changes to NvMedia Core are minimal and only reflect the state of the headers.

Unused structs and enums such as `NvMediaPoint`, `NvMediaPointFloat`, `NvMediaPointDouble`, `NvMediaColorStandard`, `NvMediaTimeBase` and `NvMediaGlobalTime` are removed.

NvMedia 2D: Migrating from 5.2 to 6.0

Table 31.   Possible Sequences for API Calls

DRIVE OS 5.2	DRIVE OS 6.0
| ```
/* Initialization */
NvMedia2D* handle =
NvMedia2DCreate(device);
NvMedia2DImageRegister(handle,
&srcImg, NVMEDIA_ACCESS_MODE_READ);
NvMedia2DImageRegister(handle,
&dstImg,
NVMEDIA_ACCESS_MODE_READ_WRITE);
``` | ```
/* Initialization */
NvMedia2DCreate(&handle, NULL);
NvMedia2DFillNvSciBufAttrList(handle,
srcBufAttrList);
NvMedia2DFillNvSciBufAttrList(handle,
dstBufAttrList);
NvMedia2DFillNvSciSyncAttrList(handle,
preSyncObjAttrsList,
NVMEDIA_PRESYNCOBJ);
NvMedia2DFillNvSciSyncAttrList(handle,
eofSyncObjAttrsList,
NVMEDIA_EOFSYNCOBJ);
NvMedia2DRegisterNvSciBufObj(handle,
srcBuf);
NvMedia2DRegisterNvSciBufObj(handle,
dstBuf);
NvMedia2DRegisterNvSciSyncObj(handle,
NVMEDIA_PRESYNCOBJ, preSyncObj);
NvMedia2DRegisterNvSciSyncObj(handle,
NVMEDIA_EOFSYNCOBJ, eofSyncObj);
``` |
| ```
/* Runtime */
NvMedia2DBlitParameters p;
p.validFields =
NVMEDIA_2D_BLIT_PARAMS_FILTER |
NVMEDIA_2D_BLIT_PARAMS_DST_TRANSFORM;
p.filter = filter;
p.dstTransform = transform;
NvMedia2DBlitEx(handle, &dstImg,
&dstRect, &srcImg, &srcRect, &p,
NULL);
``` | ```
/* Runtime */
NvMedia2DComposeParameters params;
NvMedia2DGetComposeParameters(handle,
&params);
NvMedia2DInsertPreNvSciSyncFence(handl
e, params, &preFence);
NvMedia2DSetNvSciSyncObjforEOF(handle,
params, eofSyncObj);
NvMedia2DSetSrcNvSciBufObj(handle,
params, 0, srcBuf);
NvMedia2DSetDstNvSciBufObj(handle,
params, dstBuf);
NvMedia2DSetSrcGeometry(handle,
params, 0, &srcRect, &dstRect,
transform);
NvMedia2DSetSrcFilter(handle, params,
0, filter);
NvMedia2DComposeResult result;
NvMedia2DCompose(handle, params,
&result);
NvMedia2DGetEOFNvSciSyncFence(handle,
&result, &eofFence);
``` |

DRIVE OS 5.2	DRIVE OS 6.0
``/* De-initialization */
NvMedia2DImageUnRegister(handle,
&srcImg);
NvMedia2DImageUnRegister(handle,
&dstImg);
NvMedia2DDestroyEx(handle);```	```/* De-initialization */
NvMedia2DUnregisterNvSciBufObj(handle,
srcBuf);
NvMedia2DUnregisterNvSciBufObj(handle,
dstBuf);
NvMedia2DUnregisterNvSciSyncObj(handle
, preSyncObj);
NvMedia2DUnregisterNvSciSyncObj(handle
, eofSyncObj);
NvMedia2DDestroy(handle);```

Table 32.   NvMedia 2D

Module: NvMedia 2D	
Header File and API	**Comments**

Header File:
 	
nvmedia_vpi.h
 	
Header File:
 	
nvmedia_2d.h
 	
API:
 	
NvMedia2D *
NvMedia2DCreate(
  NvMediaDevice *device
);
	Function signature is changed to:
	Header File:
	
	nvmedia_2d.h
	
	API:
	
	NvMediaStatus
	NvMedia2DCreate(
	  NvMedia2D **handle,
	  NvMedia2DAttributes const * const attr
	);
	
	The NvMedia2D* return value is changed to be delivered to caller through an out parameter, and the function now returns an error code.
	
	NvMediaDevice is no longer required to create NvMedia 2D context.
	
	A possibility to pass attributes controlling memory allocation for NvMedia 2D context is added.

Header File: nvmedia_2d.h **API:** NvMediaStatus	Function signature is changed to: **Header File:** nvmedia_2d.h

Module: NvMedia 2D	
Header File and API	**Comments**

``NvMedia2DDestroyEx(
  NvMedia2D *i2d
);```
	API:
	
	`` NvMediaStatus
	    NvMedia2DDestroy(
	      NvMedia2D *handle
	    );```
	
	The function name has changed. `Function NvMedia2DDestroy()` was deprecated in 5.2. Its name is now reused in the function replacing `NvMedia2DDestroyEx()` in 6.0.

Header File:
 	
`    nvmedia_2d.h`
 	
API:
 	
``   NvMediaStatus
    NvMedia2DImageRegister(
      const NvMedia2D *i2d,
      const NvMediaImage *image,
      NvMediaAccessMode accessMode
    );```
	`NvMediaImage` is no longer used. Instead, `NvSciBufObj` is used for image data and `NvSciSyncObj` for synchronizing NvMedia 2D operations with other tasks.
	
	`NvSciBufObjs` need to be registered with NvMedia 2D before they are used with function
	
	Header File:
	
	`    nvmedia_2d_sci.h`
	
	API:
	
	`    NvMedia2DRegisterNvSciBufObj()`
	
	The attribute lists used to create the registered `NvSciBufObjs` need to have NvMedia 2D mandatory attributes set with function
	
	Header File:
	
	`    nvmedia_2d_sci.h`
	
	API:
	
	`    NvMedia2DFillNvSciBufAttrList()`
	
	`NvSciSyncObjs` need to be registered with NvMedia 2D before they are used with function

Module: NvMedia 2D	
Header File and API	**Comments**

	Header File:
	
	    `nvmedia_2d_sci.h`
	
	API:
	
	    `NvMedia2DRegisterNvSciSyncObj()`
	
	The attribute lists used to create the registered NvSciSyncObjs need to have NvMedia 2D mandatory attributes set with function
	
	Header File:
	
	    `nvmedia_2d_sci.h`
	
	API:
	
	    `NvMedia2DFillNvSciSyncAttrList()`

Header File:
 	
    `nvmedia_2d.h`
 	
API:
 	
    `NvMediaStatus`
    `NvMedia2DImageUnRegister(`
      `const NvMedia2D *i2d,`
      `const NvMediaImage *image`
    `);`
	`NvMediaImage` is no longer used. Instead, `NvSciBufObj` is used for image data and `NvSciSyncObj` for synchronizing NvMedia 2D operations with other tasks
	
	`NvSciBufObjs` need to be unregistered with NvMedia 2D after they are no longer used with function
	
	Header File:
	
	    `nvmedia_2d_sci.h`
	
	API:
	
	    `NvMedia2DUnregisterNvSciBufObj()`
	
	`NvSciSyncObjs` need to be unregistered with NvMedia 2D after they are no longer used with function

Module: NvMedia 2D	
Header File and API	**Comments**

	Header File:
	
	    `nvmedia_2d_sci.h`
	
	API:
	
	    `NvMedia2DUnregisterNvSciSyncObj()`

Header File:
 	
    `nvmedia_2d.h`
 	
API:
 	
    `NvMediaStatus`
    `NvMedia2DBlitEx(`
      `const NvMedia2D *i2d,`
      `const NvMediaImage *dstSurface,`
      `const NvMediaRect *dstRect,`
      `const NvMediaImage *srcSurface,`
      `const NvMediaRect *srcRect,`
      `const NvMedia2DBlitParameters`
`params,`
      `NvMedia2DBlitParametersOut`
`*paramsOut`
    `);`
	`NvMediaImage` is no longer used. Instead, `NvSciBufObj` is used for image data and `NvSciSyncObj` for synchronizing NvMedia 2D operations with other tasks.
	
	The parameters of an NvMedia 2D operation are no longer passed directly to the function triggering the processing, but instead they are now configured to a separate parameters object. New parameters object needs to be acquired for each frame with function
	
	Header File:
	
	    `nvmedia_2d.h`
	
	API:
	
	    `NvMedia2DGetComposeParameters()`
	
	The source and destination surfaces are configured to the parameters object with functions
	
	Header File:
	
	    `nvmedia_2d_sci.h`
	
	API:
	
	    `NvMedia2DSetSrcNvSciBufObj()`
	    `NvMedia2DSetDstNvSciBufObj()`
	
	The source and destination rectangles, and the 2D rotation/transformation to apply are configured to the parameters object with function

Module: NvMedia 2D	
Header File and API	**Comments**

	Header File:
	nvmedia_2d.h
	
	API:
	    NvMedia2DSetSrcGeometry()
	
	The filtering is configured to the parameters object with function
	
	Header File:
	    nvmedia_2d.h
	
	API:
	    NvMedia2DSetSrcFilter()
	
	If there is a need to wait for some other task (for example another hardware engine providing input frame to NvMedia 2D in an image processing pipeline) to complete before starting the NvMedia 2D operation, a pre-fence can be configured to the parameters object with function
	
	Header File:
	    nvmedia_2d_sci.h
	
	API:
	    NvMedia2DInsertPreNvSciSyncFence()
	
	If there is a need to wait for the NvMedia 2D operation to complete, an end-of-frame fence can be requested to be generated by the operation. This is done by configuring an end-of-frame NvSciSyncObj to the parameters object with function
	
	Header File:
	    nvmedia_2d_sci.h

Module: NvMedia 2D	
Header File and API	**Comments**

	API:
	
	`NvMedia2DSetNvSciSyncObjforEOF()`
	
	The end-of-frame fence can be retrieved after triggering the operation with function
	
	Header File:
	
	`nvmedia_2d_sci.h`
	
	API:
	
	`NvMedia2DGetEOFNvSciSyncFence()`
	
	Triggering a new NvMedia 2D operation using a parameters object is done with function
	
	Header File:
	
	`nvmedia_2d.h`
	
	API:
	
	`NvMedia2DCompose()`
	
	As a new feature, there's now a possibility to use multiple source surfaces for a single NvMedia 2D operation. This is done by identifying the different surfaces with an index parameter passed to the functions with "Src" in their name.
	
	As a new feature, there's now a possibility to configure the blending mode to use with each source surface to the parameters object. This is done with function
	
	Header File:
	
	`nvmedia_2d.h`

Module: NvMedia 2D	
Header File and API	**Comments**
	API: `NvMedia2DSetSrcBlendMode()`

NvMedia Lens Distortion Correction (LDC): Migrating from 5.2 to 6.0

Table 33.   Possible Sequences for API Calls

DRIVE OS 5.2	DRIVE OS 6.0
| ```
/* Initialization */
NvMediaLDCInitParams initParams;
initParams.ldcMode =
NVMEDIA_LDC_MODE_GEOTRANS;
initParams.geoTransParams.geoTransMode
= NVMEDIA_GEOTRANS_MODE_FEED_MAPPING;
initParams.geoTransParams.filter =
filter;
initParams.geoTransParams.bitMaskEnabl
e = NVMEDIA_TRUE;
initParams.geoTransParams.bitMaskMap =
maskMap;
NvMediaLDCCreateNew(
  device,
  &handle,
  srcW,
  srcH,
  &srcRect,
  dstW,
  dstH,
  &dstRect,
  &initParams
);
NvMediaLDCFeedSparseWarpMap(handle,
&warpMap);
``` | ```
/* Initialization */
NvMediaLdcCreate(&handle, NULL);
NvMediaLdcFillNvSciBufAttrList(handle,
srcBufAttrList);
NvMediaLdcFillNvSciBufAttrList(handle,
dstBufAttrList);
NvMediaLdcFillNvSciBufAttrList(handle,
xsobelDstBufAttrList);
NvMediaLdcFillNvSciBufAttrList(handle,
xsobelDsDstBufAttrList);
NvMediaLdcFillNvSciSyncAttrList(handle
, preSyncObjAttrsList,
NVMEDIA_PRESYNCOBJ);
NvMediaLdcFillNvSciSyncAttrList(handle
, eofSyncObjAttrsList,
NVMEDIA_EOFSYNCOBJ);
NvMediaLdcRegisterNvSciBufObj(handle,
srcBuf);
NvMediaLdcRegisterNvSciBufObj(handle,
dstBuf);
NvMediaLdcRegisterNvSciBufObj(handle,
xsobelDstBuf);
NvMediaLdcRegisterNvSciBufObj(handle,
xsobelDsDstBuf);
NvMediaLdcRegisterNvSciSyncObj(handle,
NVMEDIA_PRESYNCOBJ, preSyncObj);
``` |

DRIVE OS 5.2	DRIVE OS 6.0
| | ```
NvMediaLdcRegisterNvSciSyncObj(handle,
NVMEDIA_EOFSYNCOBJ, eofSyncObj);
NvMediaLdcParameters params;
NvMediaLdcCreateParameters(handle,
&paramsAttrs, &params);
NvMediaLdcSetNvSciSyncObjforEOF(handle
, params, eofSyncObj);
NvMediaLdcSetFilter(handle, params,
filter);
NvMediaLdcSetGeometry(handle, params,
&srcRect, &dstRect);
NvMediaLdcSetWarpMapParameters(handle,
params, &warpMapParams);
NvMediaLdcSetMaskMapParameters(handle,
params, &maskMapParams);
``` |
| ```
/* Runtime */
NvMediaLDCCtrlParams ctrlParams;
ctrlParams.xSobelMode =
NVMEDIA_GEOTRANS_ENABLE_XSOBEL_ENABLE_
DS;
NvMediaLDCProcess(handle, NULL,
srcImg, dstImg, xsobelImg,
xsobelDsImg, &ctrlParams);
``` | ```
/* Runtime */
NvMediaLdcInsertPreNvSciSyncFence(hand
le, params, preFence);
NvMediaLdcSetSrcNvSciBufObj(handle,
params, srcBuf);
NvMediaLdcSetDstNvSciBufObj(handle,
params, dstBuf);
NvMediaLdcSetXSobelDstSurface(handle,
params, xsobelDstBuf);
NvMediaLdcSetDownsampledXSobelDstSurfa
ce(handle, params, xsobelDsDstBuf);
NvMediaLdcResult result;
NvMediaLdcProcess(handle, params,
&result);
NvMediaLdcGetEOFNvSciSyncFence(handle,
&result, &eofFence);
``` |
| ```
/* De-initialization */
NvMediaLDCDestroy(handle);
``` | ```
/* De-initialization */
NvMediaLdcDestroyParameters(handle,
params);
NvMediaLdcUnregisterNvSciBufObj(handle
, srcBuf);
NvMediaLdcUnregisterNvSciBufObj(handle
, dstBuf);
NvMediaLdcUnregisterNvSciBufObj(handle
, xsobelDstBuf);
NvMediaLdcUnregisterNvSciBufObj(handle
, xsobelDsDstBuf);
NvMediaLdcUnregisterNvSciSyncObj(handl
e, preSyncObj);
NvMediaLdcUnregisterNvSciSyncObj(handl
e, eofSyncObj);
NvMediaLdcDestroy(handle);
``` |

Table 34.   NvMedia Lens Distortion Correction

Module: NvMedia LDC	
Header File and API	**Comments**

Header File:
 	
   `nvmedia_ldc.h`
 	
API:
 	
  `NvMediaStatus`
  `NvMediaLDCCreateNew(`
    `const NvMediaDevice *device,`
    `NvMediaLDC **pldc,`
    `const uint16_t srcWidth,`
    `const uint16_t srcHeight,`
    `const NvMediaRect *srcRect,`
    `const uint16_t dstWidth,`
    `const uint16_t dstHeight,`
    `const NvMediaRect *dstRect,`
    `const NvMediaLDCInitParams *initParams`
  `);`
	NvMediaDevice is no longer required to create NvMedia LDC context. The context is created with function
	
	Header File:
	
	  `nvmedia_ldc.h`
	
	API:
	
	  `NvMediaStatus`
	  `NvMediaLdcCreate(`
	    `NvMediaLdc **const handle,`
	    `NvMediaLdcAttributes const *const attr`
	  `);`
	
	The `NvMediaLdcAttributes` parameter controls the amount of memory allocated for the context.
	
	The surface dimensions, rectangles, filtering, regions, matrix coefficients etc. are no longer passed to the context creation function. Instead, they are now configured to a separate parameters object created with the function
	
	Header File:
	
	  `nvmedia_ldc.h`
	
	API:
	
	  `NvMediaStatus`
	  `NvMediaLdcCreateParameters(`
	    `NvMediaLdc *const handle,`
	    `NvMediaLdcParametersAttributes const *const attr,`
	    `NvMediaLdcParameters *const params`
	  `);`
	
	The `NvMediaLdcParametersAttributes` parameter controls which features (such as TNR3 and mask map.) can be used with the allocated

Module: NvMedia LDC	
Header File and API	**Comments**
	parameters object and how much memory is allocated for them.
	The pixel interpolation filter is configured to the parameters object with function
	Header File: `nvmedia_ldc.h`
	API: `NvMediaLdcSetFilter()`
	The rectangles are configured to the parameters object with function
	Header File: `nvmedia_ldc.h`
	API: `NvMediaLdcSetGeometry()`
	The IPT (perspective) matrix and related region configuration are configured to the parameters object with function
	Header File: `nvmedia_ldc.h`
	API: `NvMediaLdcSetIptParameters()`
	The warp map and related region configuration are configured to the parameters object with function

Module: NvMedia LDC	
Header File and API	**Comments**

	Header File:
	
	    `nvmedia_ldc.h`
	
	API:
	
	    `NvMediaLdcSetWarpMapParameters()`
	
	The mask map is configured to the parameters object with function
	
	Header File:
	
	    `nvmedia_ldc.h`
	
	API:
	
	    `NvMediaLdcSetMaskMapParameters()`
	
	The TNR3 parameters are configured to the parameters object with function
	
	Header File:
	
	    `nvmedia_ldc.h`
	
	API:
	
	    `NvMediaLdcSetTnrParameters()`

Header File:
 	
    `nvmedia_ldc.h`
 	
API:
 	
    `NvMediaStatus`
    `NvMediaLDCDestroy(`
      `NvMediaLDC *ldc`
    `);`
	Any created parameters objects need to be destroyed with function
	
	Header File:
	
	    `nvmedia_ldc.h`
	
	API:
	
	    `NvMediaLdcDestroyParameters()`

Module: NvMedia LDC	
Header File and API	**Comments**

	Before destroying the NvMedia LDC context. The context is destroyed with function
	
	Header File:
	
	    nvmedia_ldc.h
	
	API:
	
	    NvMediaLdcDestroy();

Header File:
 	
    nvmedia_ldc.h
 	
API:
 	
    NvMediaStatus
    NvMediaLDCFeedSparseWarpMap(
      NvMediaLDC *ldc,
      const NvMediaLDCSparseWarpMap *map
    );
	The warp map is now configured to a parameters object with function
	
	Header File:
	
	    nvmedia_ldc.h
	
	API:
	
	    NvMediaLdcSetWarpMapParameters()
	
	Only floating point format is supported for the map control points.

Header File:
 	
    nvmedia_ldc.h
 	
API:
 	
    NvMediaStatus
    NvMediaLDCMappingGen(
      NvMediaLDC *ldc
    );
	Generating a warp map based on a lens model is now done with function
	
	Header File:
	
	    nvmedia_ldc_util.h
	
	API:
	
	    NvMediaLdcGenWarpMap()

Header File: nvmedia_ldc.h	TNR2 is no longer supported.

Module: NvMedia LDC	
Header File and API	**Comments**
API: ```NvMediaStatus NvMediaLDCUpdateTNR2Params(NvMediaLDC *ldc, const NvMediaTNR2Params *tnr2Params);```	

Header File:
 	
`nvmedia_ldc.h`
 	
API:
 	
```NvMediaStatus NvMediaLDCUpdateTNR3Params(const NvMediaLDC *ldc, const NvMediaTNR3Params *tnr3Params);```
	The TNR3 parameters are configured to a parameters object with function
	
	Header File:
	
	`nvmedia_ldc.h`
	
	API:
	
	`NvMediaLdcSetTnrParameters()`

Header File:
 	
`nvmedia_ldc.h`
 	
API:
 	
```NvMediaStatus NvMediaLDCProcess(const NvMediaLDC *ldc, NvMediaImage *prevSurface, NvMediaImage *curSurface, NvMediaImage *outputSurface, NvMediaImage *xSobel, NvMediaImage *downSample, const NvMediaLDCCtrlParams *ldcCtrlParams);```
	`NvMediaImage` is no longer used. Instead, `NvSciBufObj` is used for image data and `NvSciSyncObj` for synchronizing NvMedia LDC operations with other tasks.
	
	`NvSciBufObj`s need to be registered with NvMedia LDC before they are used with function
	
	Header File:
	
	`nvmedia_ldc_sci.h`
	
	API:
	
	`NvMediaLdcRegisterNvSciBufObj()`
	
	The attribute lists used to create the registered NvSciBufObjs need to have NvMedia LDC mandatory attributes set with function
	
	Header File:
	
	`nvmedia_ldc_sci.h`

Module: NvMedia LDC	
Header File and API	**Comments**

	API:
	
	    `NvMediaLdcFillNvSciBufAttrList()`
	
	`NvSciSyncObjs` need to be registered with NvMedia LDC before they are used with function
	
	Header File:
	
	    `nvmedia_ldc_sci.h`
	
	API:
	
	    `NvMediaLdcRegisterNvSciSyncObj()`
	
	The attribute lists used to create the registered `NvSciSyncObjs` need to have NvMedia LDC mandatory attributes set with function
	
	Header File:
	
	    `nvmedia_ldc_sci.h`
	
	API:
	
	    `NvMediaLdcFillNvSciSyncAttrList()`
	
	The surfaces are no longer passed directly to the function triggering the processing of a frame, but instead they are now configured to a separate parameters object. The surfaces are configured to the parameters object with functions
	
	Header File:
	
	    `nvmedia_ldc_sci.h`
	
	API:
	
	    `NvMediaLdcSetSrcSurface()`
	    `NvMediaLdcSetDstSurface()`
	    `NvMediaLdcSetPreviousSurface()`
	    `NvMediaLdcSetXSobelDstSurface()`

Module: NvMedia LDC	
Header File and API	**Comments**

	`NvMediaLdcSetDownsampledXSobelDstSurface()`
	
	No separate parameter controls the XSobel working mode, but instead it is determined based on the surfaces configured for an operation. With 6.0.9, XSobel surface validation is more strict than with 5.2, and the surface format and dimensions need to match. See `NvMediaLdcSetXSobelDstSurface()` and `NvMediaLdcSetDownsampledXSobelDstSurface()` in the API reference documentation for additional information.
	
	If there is a need to wait for another task (for example another HW engine providing input frame to NvMedia LDC in an image processing pipeline) to complete before starting the NvMedia LDC operation, a pre-fence can be configured to the parameters object with function
	
	Header File:
	
	    `nvmedia_ldc_sci.h`
	
	API:
	
	    `NvMediaLdcInsertPreNvSciSyncFence()`
	
	If there is a need to wait for the NvMedia LDC operation to complete, an end-of-frame fence can be requested to be generated by the operation. This is done by configuring an end-of-frame NvSciSyncObj to the parameters object with function
	
	Header File:
	
	    `nvmedia_ldc_sci.h`
	
	API:
	
	    `NvMediaLdcSetNvSciSyncObjforEOF()`

Module: NvMedia LDC	
Header File and API	**Comments**
	The end-of-frame fence can be retrieved after triggering the operation with function
	Header File: nvmedia_ldc_sci.h
	API: NvMediaLdcGetEOFNvSciSyncFence()
	The same parameters object (with same warp map, TNR3 etc. configuration) can be used for multiple frames, but the surfaces and pre-fences need to be configured separately for each frame.
	Triggering a new NvMedia LDC operation using a parameters object is done with function
	Header File: nvmedia_ldc.h
	API: NvMediaLdcProcess()
	NvSciBufObjs need to be unregistered with NvMedia LDC after they are no longer used with function
	Header File: nvmedia_ldc_sci.h
	API: NvMediaLdcUnregisterNvSciBufObj()
	NvSciSyncObjs need to be unregistered with NvMedia LDC after they are no longer used with function

Module: NvMedia LDC	
Header File and API	**Comments**
	Header File: nvmedia_ldc_sci.h **API:** NvMediaLdcUnregisterNvSciSyncObj()

Camera SIPL GPIO and Interrupt Localization Changes Introduced in DRIVE OS 6.0.6.0

Camera SIPL GPIO control is now consolidated in the CDAC (6.0.7.0 and prior) or CamGPIO Resource Managers on QNX, and cdi-mgr on Linux. All Camera GPIO functionalities should be accessed through the SIPL CDI GPIO APIs from device drivers (CDD):

> DevBlkCDIRootDeviceSetGPIOPinLevel()
> DevBlkCDIRootDeviceGetGPIOPinLevel()
> DevBlkCDIRootDeviceCheckAndClearIntr()*
> DevBlkCDIRootDeviceWaitForError()*
> DevBlkCDIRootDeviceAbortWaitForError()*

(The recommendation for CDD drivers is to use asynchronous SIPL interrupt localization rather than handling interrupts directly.)

Interrupt Localization

Camera error interrupt localization from external hardware sources is introduced in Camera SIPL for Orin in 6.0.6.0. Each camera group has a pair of falling- and rising-edge triggered interrupt lines to Tegra, to which all corresponding interrupt signals in all devices within that group are muxed together.

SIPL Interrupt Localization is a Device Driver Interface (DDI) implemented in device drivers (CDD) triggered and supervised by SIPL Core upon receiving an interrupt, to determine the origins of a muxed interrupt signal, and to propagate the appropriate error interrupt event signals to the client within a time limit (such as FTTI). An optional timeout value is configurable in the SIPL Platform Configuration. A timeout error notification is dispatched to the client if error localization is not completed within the duration.

Interrupt localization is enabled by setting the `enableGetStatus` flag in the SIPL Platform Configuration for the GPIO pin. If not enabled, interrupts are propagated to the client notification queue immediately, which was the legacy behavior.

Device Tree Configuration

Under the new GPIO regime in Orin, all Tegra GPIO pins used by camera hardware should be configured in the Device Tree under the new `/sipl_devblk_X/tegra/gpios` node (X is the I2C bus number of the Camera Group).

Table 35.   Device Tree Configuration

Item	Format	Description
index	Non-negative integer unique in this scope	Identifier unique in the scope of a Camera Group, used for matching with the SIPL Platform Configuration.
nvgpio-line	phandle to NvGpio Device Tree Node	Reference to the underlying NvGpio node.
Interrupt type	Any single instance of {`input`, `output`, `intr-level-low`, `intr-level-high`, `intr-edge-any`, `intr-edge-rising`, `intr-edge-falling`}	Type of Camera Interrupt, must be compatible with the NvGPIO Device Tree node configuration (checked at CamGPIO initialization at system boot).

Example

By default, the falling- and rising-edge triggered interrupts are present in the device tree.

```
# Under `/`
sipl_devblk_0 {
    tegra {
        gpios {
            DEVBLK0_DSER_CAM_PWR_SHARED_GPIO: 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;
            };
        };
```

```
    };
};
```

Platform Configuration

The SIPL Platform Configuration is extended to support interrupt localization and interrupt localization timeout. The `errGpios` attribute accepts a list of Camera GPIO interrupt GPIOs to monitor.

Table 36.   Platform Configuration

Attribute	Format	Description
idx	Non-negative integer unique in this scope	Identifier unique in the scope of a Camera Group, used for matching with the SIPL Device Tree configuration.
enableGetStatus	True or False	Interrupt Localization enablement. Default: false
timeout_ms	Non-negative integer	Interrupt Localization timeout in milliseconds, no effect if `enableGetStatus` is false. Default: 0 (disabled).

Example

```
"deserializers": [
    {
        ...
     "errGpios": [
        { "idx": "0", "enableGetStatus": true, "timeout_ms": 0 }
     ],
     "pwrGpio": ["7"],
    },
    ...
}
```

SIPL Event Deprecation

With NVIDIA DRIVE OS 6.0.8.1, SIPL will no longer send the following pipeline events:

> `NOTIF_INFO_ISP_PROCESSING_DONE`

> `NOTIF_INFO_ACP_PROCESSING_DONE`

These events are marked as deprecated in the SIPL header and will be removed in future releases. To avoid compilation failure, do not use these APIs.

NvStreams API Changes

About NvStreams

NvStreams is a family of technologies related to communicating information between different interconnected hardware domains and software partitions. It provides software with the flexibility in constructing different execution pipelines between applications independent of actual system configuration and topology. NvStreams implements three specific technologies: NvSciBuf, NvSciSync, and NvSciStream.

This section summarizes the API changes that users of NvStreams can expect to see moving from NVIDIA DRIVE OS 5.2 to 6.0, keeping in mind that NVIDIA DRIVE 6.0 is still in development and functionality is subject to change.

About the NvStreams Migration

The API changes fall into a few general categories, briefly described here. The details and examples of how to make the transition, are in the following sections.

In 5.x, our approach for transmitting data through the stream was that every time data was provided on one block, it was immediately transmitted through the stream to the other blocks, where it would arrive as an event. Each type of data had its own event type, which applications needed to recognize and know how to handle. Starting with 6.0, we will organize different types of data into groups. Data will be held in the block where it is specified until the user indicates it is done with the relevant group. Then the date will be transmitted through the stream at once, resulting in a single event at each recipient. The recipient can then query and handle the data at a time of its choosing, possibly restricting its queries to a subset of the data it needs. This allows applications to organize themselves into different setup operations, and smoothly transition to streaming when setup is complete. It also allows for future expansion of NvSciStream to support optional features.

In 5.x, the producer and consumer(s) each provided a global list of sync objects to coordinate their use of the buffers. The intent was for different engines, which operated on separate buffers, to use different sync objects. Nothing in the API, however, made it clear which sync objects were relevant to the data in each buffer or engine. In 6.0, rather than specifying a global list of sync objects, endpoints will specify a single sync object for each element in the packets.

In 6.0, when IPC blocks are destroyed, they will also close associated channels. Previously, the channels were left active, and applications that were fully cleaned up would have to close them themselves. These applications must be updated or they will free the channels twice.

6.0 will introduce error codes to allow for easier debugging if anything goes wrong. Any applications that check for specific error codes will need to be updated to include the new ones.

NvStreams Examples

For most of the changed APIs, examples of how to perform the transition are provided. The examples assume a relatively simple case with two packet elements: one accessed synchronously, and one accessed asynchronously. In this code, all consumers use both elements, but comments are provided to describe how they might take advantage of the new APIs if they only access a subset. The following symbols are used to name the two elements:

```
#define MY_ELEMENT_TYPE_BASE  0x1234
#define MY_ELEMENT_TYPE_SYNC  (MY_ELEMENT_TYPE_BASE+0)
#define MY_ELEMENT_TYPE_ASYNC (MY_ELEMENT_TYPE_BASE+1)
```

For many operations, the producer and consumer code would be very similar. In these cases, to avoid duplicating large blocks of code with minor differences, only the producer version is shown, and the differences for the consumer are indicated by comments.

For simplicity, error handling has been omitted from most of the examples.

Application-specific operations to generate data to pass to NvSciStream and process data received from it is referenced by a function of the form `Application_SomeOperation();`

General Changes

Some of the new functions will require the producer application to iterate over the list of consumers. Simple applications with fixed configurations may already know the number of consumers. For more generally configurable applications, this information will now be available through the producer and pool blocks once the stream is fully connected via a new query:

```
NvSciError
NvSciStreamBlockConsumerCountGet(
    NvSciStreamBlock const block,
    uint32_t* const numConsumers);
```

For the functions where iteration over consumers is performed, the queries will include a parameter to specify the block index. When querying information about consumers, this index should be that of the consumer. When querying information about producers, this index should always be 0.

```
SomeFunction( … , uint32_t const queryBlockIndex, … );
```

Some queries on the pool block can obtain information about either the producer or consumer(s). These functions include a new parameter to specify the source of the information being queried

```
typedef enum {
    NvSciStreamBlockType_Producer,
    NvSciStreamBlockType_Consumer,
    NvSciStreamBlockType_Pool
} NvSciStreamBlockType;

SomeFunction( … , NvSciStreamBlockType const queryBlockType, … );
```

Various setup operations will be broken up into key groups. Rather than transmitting every piece of data through the stream as it arrives, required and optional data for each group is gathered and sent all at once when the application indicates that it is done with the setup. This completion is signaled with a new function with the use described further in the following sections. The `<completed>` parameter to this function is provided for future support for dynamically modifying streams (outside of safety systems) and currently must always be `true`.

```
typedef enum { … } NvSciStreamSetup;

NvSciError
NvSciStreamBlockSetupStatusSet(
    NvSciStreamBlock const block,
    NvSciStreamSetup const setupType,
    bool const completed);
```

IPC Setup

Applications that stream between processes must call `NvSciIpcResetEndpointSafe()` on any channel endpoint they open before using it to create an NvSciStream IpcSrc or IpcDst block, regardless of whether they perform communication over the channel before passing ownership.

Connection

Creation and connection of the stream is performed as in 5.2. After the stream is fully connected, the producer and pool can query the number of consumers, as described above.

Example

Unmodified Code

```
//
// All block creation and connection functions are as in 5.2
//
…
```

New Code

```
// In producer process:
//   After connection, query the number of consumers.
//   This is not necessary for simple applications where the producer
//   always knows how many consumers there will be.
uint32_t numConsumers;
NvSciStreamBlockConsumerCountGet(producerBlock, &numConsumers);
```

Event Handling

The event query function will no longer return an event structure. Instead, it will just return an event type. The data previously returned in this structure will be available to query at any time after the event is received using new functions described below. Applications receiving events can process the event data in any order they choose. Events will also be consolidated, so a single event will be received for groups of related data.

```
NvSciError
NvSciStreamBlockEventQuery(
    NvSciStreamBlock const block,
    int64_t const timeoutUsec,
    NvSciStreamEvent *const event
    NvSciStreamEventType *const eventType);
```

Motivation for Change: With the previous model, we required separate events for every piece of information received at a block. Adding new features, particularly optional ones, becomes difficult because we must add new event types, requiring application updates to recognize, and potentially grow, the data structure, requiring recompilation of applications even if they do not need to support the new event. The new model reduces the number of events to a smaller set of events indicating various types of data are ready. New features simply add to the set of data available, but older applications, which are not interested in the new data, will not have to be modified or recompiled to accommodate changes in the data structure.

Example

This example illustrates the transition required for a generic (non-existent) array of "Foo" data. Specific examples are provided for each real data type. For brevity, those examples omit the outer loop and just show the case statement in the event switch.

Old Code

```
// Main event loop
while (<not done>) {
    …
    NvSciStreamEvent event;
    if (NvSciError_Success !=
        NvSciStreamBlockEventQuery(block, timeout, &event)) {
        switch (event.type) {
        …
        case NvSciStreamEventType_FooCount:
            // Handle the count received in the event
            Application_FooArraySetup(event.count);
            break;
        case NvSciStreamEventType_SingleFoo:
            // Handle the individual indexed foo received in the event
            Application_FooHandler(event.index, event.fooValue);
            break;
        …
        }
    }
    …
}
```

New Code

```
// Main event loop
while (<not done>) {
    …
    NvSciStreamEventType eventType;
    if (NvSciError_Success !=
        NvSciStreamBlockEventQuery(block, timeout, &eventType)) {
        switch (eventType) {
        …
        case NvSciStreamEventType_AllFoos:
            // Query and handle the count
            uint32_t fooCount;
            NvSciStreamBlockFooCountGet(block, &fooCount);
            Application_FooArraySetup(fooCount);
            // Query and handle each indexed value
            for (uint32_t j; j<fooCount; j++) {
                FooType fooValue;
```

```
            NvSciStreamBlockFooValueGet(block, j, &fooValue);
            Application_FooHandler(j, fooValue);
        }
        break;
    …
    }
  }
  …
}
```

Error Events

If an error event (`NvSciStreamEventType_Error`) occurs on any block, the associated error value can be queried with a new function:

```
NvSciError
NvSciStreamBlockErrorGet(
    NvSciStreamBlock const block,
    NvSciError* const status);
```

Element Support

Specifying Element Support

There will be a minor change in how the producer and consumer(s) specify the types of packet elements they support, and how the pool specifies the final element layout. Instead of specifying the number of elements up front and then each element by index, the elements will be passed without an index. After specifying all of the elements, the application will indicate it is done with the element setup by calling `NvSciStreamBlockSetupStatusSet()` with a value of `NvSciStreamSetup_ElementExport`. The indices and count will be determined automatically by NvSciStream.

Motivation for Change: Elimination of the count and index provides a more natural means of specifying the support, particularly for simple applications that only provide a single element. The addition of the completion signal after the elements are specified also allows for future features that specify optional support along with the list of elements.

The synchronization mode parameter will also be eliminated from the element specification. Instead, this will be covered by the specification of synchronization object attributes, described below.

Motivation for Change: The original interfaces had redundant and potentially conflicting, methods for specifying whether data is written and read synchronously or asynchronously. We are consolidating on a single method to eliminate confusion.

```
NvSciError
NvSciStreamBlockPacketElementCount(
    NvSciStreamBlock const block,
    uint32_t const count);

NvSciError
NvSciStreamBlockPacketAttr(
NvSciStreamBlockElementAttrSet(
    NvSciStreamBlock const block,
    uint32_t const index,
    uint32_t const userType,
    NvSciStreamElementMode const syncMode,
    NvSciBufAttrList const bufAttrList);
```

Producer (and Consumer) Example

Old Code

```
// Set up the attributes for each buffer type we generate
//   (For consumers, obtain read attributes instead of write)
NvSciBufAttrList elemAttrs[2];
Application_ElementWriteAttrs(elemAttrs);
// Inform NvSciStream there are 2 elements and send each of them
NvSciStreamBlockPacketElementCount(block, 2);
NvSciStreamBlockPacketAttr(block, 0, MY_ELEMENT_TYPE_SYNC,
                           NvSciStreamElementMode_Immediate,
                           elemAttrs[0]);
NvSciStreamBlockPacketAttr(block, 1, MY_ELEMENT_TYPE_ASYNC,
                           NvSciStreamElementMode_Asynchronous,
                           elemAttrs[1]);
```

New Code

```
// Set up the attributes for each buffer type we generate
//   (For consumers, obtain read attributes instead of write)
NvSciBufAttrList elemAttrs[2];
Application_ElementWriteAttrs(elemAttrs);
// Inform NvSciStream of each element
//   (If a consumer doesn't need one of these, it just skips it)
NvSciStreamBlockElementAttrSet(block, MY_ELEMENT_TYPE_SYNC,
                               elemAttrs[0]);
NvSciStreamBlockElementAttrSet(block, MY_ELEMENT_TYPE_ASYNC,
                               elemAttrs[1]);
// Inform NvSciStream that element specification is done
NvSciStreamBlockSetupStatusSet(block,
                               NvSciStreamSetup_ElementExport,
                               true);
```

Pool Example

Refer to the next section for the combined receive and send example.

Receiving Element Support

Previously, the number of elements and individual element types arrived as separate events on each block. Now, the pool, producer, and consumer will each receive a single `NvSciStreamEvent_Elements` event. The pool will receive this event after the producer and all consumers finish specifying their element support. After the pool has finished specifying the final packet element layout, the producer and consumers receive this event.

After receiving this event, the application queries the element information using new functions instead of obtaining it as data in the event structure. The pool should use a `<queryBlockType>` of `NvSciStreamBlockType_Producer` or `NvSciStreamBlockType_Consumer` to get the information from the producer or consumers, respectively. The producer and consumer should use a `<queryBlockType>` of `NvSciStreamBlockType_Pool`.

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 the new `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_ElementImport`. This allows them to begin receiving packets from the pool.

```
typedef enum {
    …
    NvSciStreamEventType_PacketElementCountProducer
    NvSciStreamEventType_PacketElementCountConsumer
    NvSciStreamEventType_PacketElementCount
    NvSciStreamEventType_PacketAttrProducer
    NvSciStreamEventType_PacketAttrConsumer
    NvSciStreamEventType_PacketAttr
    NvSciStreamEventType_Elements

    …
} NvSciStreamEventType;

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);

    NvSciBufAttrList* const bufAttrList);

NvSciError
NvSciStreamBlockElementUsageSet(
    NvSciStreamBlock const block,
    uint32_t const elemIndex,
    bool const used);
```

Pool Example

For simplicity, this code assumes that the producer and consumer express support for both element types. A more generic pool application would use the received counts, check the list of support indicated, and decide the subset of types to use for the final packet layout.

Old Pool Code

```
uint32_t prodElemFound = 0, consElemFound = 0;
NvSciBufAttrList supportElemAttr[2][2], packetElemAttr[2];
bool elemSetupDone = false;
…
case NvSciStreamEventType_PacketElementCountProducer:
case NvSciStreamEventType_PacketElementCountConsumer:
    // For this simple example, expect count is always 2
    assert(event.count == 2);
    break;
case NvSciStreamEventType_PacketAttrProducer:
    // Save incoming producer attribute list
    myIndex = event.userData - MY_ELEMENT_TYPE_BASE;
    supportElemAttr[myIndex][0] = event.bufAttrList;
    prodElemFound++;
    break;
case NvSciStreamEventType_PacketAttrConsumer:
    // Save incoming consumer attribute list
    myIndex = event.userData - MY_ELEMENT_TYPE_BASE;
    supportElemAttr[myIndex][1] = event.bufAttrList;
```

```
        consElemFound++;
        break;
…
// After event-handling switch
…
// When all expected element attributes are received, set up the
//   final packet layout
if (!elemSetupDone && (prodElemFound == 2) && (consElemFound == 2)) {
    // Combine producer and consumer attributes
    for (uint32_t j=0; j<3; ++j) {
        // Reconcile attributes
        NvSciBufAttrList conflicts;
        NvSciBufAttrListReconcile(supportElemAttr[j], 2,
                                  &packetElemAttr[j], &conflicts);
    }
    // Specify final layout
    NvSciStreamBlockPacketElementCount(pool, 2);
    NvSciStreamBlockPacketAttr(pool, 0, MY_ELEMENT_TYPE_SYNC,
                               NvSciStreamElementMode_Immediate,
                               packetElemAttr[0]);
    NvSciStreamBlockPacketAttr(pool, 1, MY_ELEMENT_TYPE_ASYNC,
                               NvSciStreamElementMode_Asynchronous,
                               packetElemAttr[1]);
    // Mark setup done so we don't do this again
    elemSetupDone = true;
}
```

New Pool Code

```
NvSciBufAttrList packetElemAttr[2];
…
// Handle all incoming element descriptions at once
case NvSciStreamEventType_Elements:
    // For this simple example, the counts are known to always be 2,
    //   so the queries could be skipped. They are done here solely to
    //   illustrate how to do them.
    uint32_t elemCount;
    NvSciStreamBlockElementCountGet(pool,
                                    NvSciStreamBlockType_Producer,
                                    &elemCount);
    assert(elemCount == 2);
    NvSciStreamBlockElementCountGet(pool,
                                    NvSciStreamBlockType_Consumer,
                                    &elemCount);
    assert(elemCount == 2);

    // Query the buffer attribute lists from both endpoints
    NvSciBufAttrList supportElemAttr[2][2];
```

```
    for (uint32_t j=0; j<2; ++j) {
        uint32_t userType;
        NvSciBufAttrList attr;
        NvSciStreamBlockElementAttrGet(pool,
                                        NvSciStreamBlockType_Producer,
                                        j, &userType, &attr);
        supportElemAttr[userType-MY_ELEMENT_TYPE_BASE][0] = attr;
        NvSciStreamBlockElementAttrGet(pool,
                                        NvSciStreamBlockType_Consumer,
                                        j, &userType, &attr);
        supportElemAttr[userType-MY_ELEMENT_TYPE_BASE][1] = attr;
    }
    // Inform NvSciStream that element import is done
    NvSciStreamBlockSetupStatusSet(pool,
                                    NvSciStreamSetup_ElementImport,
                                    true);

    // Combine and send producer and consumer attributes
    for (uint32_t j=0; j<2; ++j) {
        NvSciBufAttrList conflicts;
        NvSciBufAttrListReconcile(supportElemAttr[j], 2,
                                  &packetElemAttr[j], &conflicts);
        NvSciStreamBlockElementAttrSet(pool,
                                        MY_ELEMENT_TYPE_BASE+j,
                                        packetElemAttr[j]);
    }
    // Inform NvSciStream that element specification is done
    NvSciStreamBlockSetupStatusSet(pool,
                                    NvSciStreamSetup_ElementExport,
                                    true);
    break;
…
```

Producer and Consumer Example

> **Note:** In this and all subsequent examples, we assume that the pool has specified the elements in an expected order, with the synchronous element first and the asynchronous one second. In a more general application, where the set of elements isn't fixed, the producer and consumer need to keep track of the indices for each type for use when querying sync and buffer data.

Old Producer/Consumer Code

```
NvSciBufAttrList packetElemAttr[2];
…
case NvSciStreamEventType_PacketElementCount:
```

```
    // For this simple example, expect count is always 2
    assert(event.count == 2);
    break;
case NvSciStreamEventType_PacketAttr:
    // Save the element info
    myIndex = event.userData - MY_ELEMENT_TYPE_BASE;
    assert(myIndex == event.index);
    packetElemAttr[myIndex] = event.bufAttrList;
    break;
…
```

New Producer/Consumer Code

```
NvSciBufAttrList packetElemAttr[2];
…
case NvSciStreamEventType_Elements:
    // For this simple example, the count is known to always be 2,
    //   so the query could be skipped. It is done here solely to
    //   illustrate how to do it.
    uint32_t elemCount;
    NvSciStreamBlockElementCountGet(block,
                                    NvSciStreamBlockType_Pool,
                                    &elemCount);
    assert(elemCount == 2);

    // Record the element info
    for (uint32_t j=0; j<2; ++j) {
        uint32_t userType;
        NvSciBufAttrList attr;
        NvSciStreamBlockElementAttrGet(block,
                                       NvSciStreamBlockType_Pool,
                                       j, &userType, &attr);
        // The consumer can skip any elements which it doesn't use by
        //   adding the following call. This will not be needed in most
        //   existing applications, and is intentionally commented out here.
        // NvSciStreamBlockElementUsageSet(block, j, false);
        myIndex = userType – MY_ELEMENT_TYPE_BASE;
        assert(myIndex == j);
        packetElemAttr[myIndex] = attr;
    }

    // Inform NvSciStream that element import is done
    NvSciStreamBlockSetupStatusSet(block,
                                   NvSciStreamSetup_ElementImport,
                                   true);    break;
…
```

Element Sync Attributes

Instead of a list of global sync objects, sync objects will be specified on a per element basis. (It will still be possible to use the same sync object with multiple elements.) Because of this, providing the sync attributes occurs after the endpoints receive the packet layout information. Producers are required to generate data for all elements.

Motivation for Change: Having a list of multiple sync objects was intended to support the case where the contents of some buffers were written or read by different engines operating independently, each signaling their fence when they are done. However, because the sync objects were defined globally, there was nothing to indicate for which buffers each one was relevant. The other endpoint(s) would have to wait for all the fences, even if only one of them pertained to the data they cared about. Per-element sync objects allow the original intention to be achieved.

The `NvSciStreamBlockSyncRequirements()` function will be replaced with a new function that provides the endpoint's waiter requirements for that element. The `<synchronousOnly>` parameter is eliminated. If the endpoint requires the data to be generated synchronously, it can pass `NULL` for the attribute list, or it can omit the call and NULL will be assumed.

If the consumer will not use an element, it is not necessary for it to provide a corresponding waiter attribute list, but it is important that it informs NvSciStream that it will not use the element, as described in the previous section. Otherwise, the producer will be told that sync objects are not supported for the element even if the consumers that use it provide attribute lists.

```
NvSciError
NvSciStreamBlockSyncRequirements(
NvSciStreamBlockElementWaiterAttrSet(
    NvSciStreamBlock const block,
    uint32_t const elemIndex,
    bool const synchronousOnly,
    NvSciSyncAttrList const waitSyncAttrList);
```

Example

This example just takes the sync attributes previously sent globally and associates them with the single element used asynchronously. An application that uses more than one element generated by the same engine can pass the same attribute list for each element, while one that uses elements generated by different engines can pass separate attribute lists for each element.

Old Code

```
// Obtain waiter attributes for engine used for asynchronous element
NvSciSyncAttrList waiterAttr;
Application_GetEngineWaitAttrs(&waiterAttr);
```

```
// Send attributes
NvSciStreamBlockSyncRequirements(block, false, waiterAttr);
```

New Code

In practice, this code would probably be merged with that from the previous section, where the list of packet elements is queried. An application would specify which elements it uses, and the sync attributes for them as it receives the elements.

```
// Note that if a consumer doesn't use one of the elements, it would
//   skip the call that indicates support for it

// Indicate that the synchronous element will be used without any
//   sync object by passing NULL for the attributes
// This call could be omitted since NULL is the default

NvSciStreamBlockElementWaiterAttrSet(block, 0, NULL);

// Obtain waiter attributes for engine used for asynchronous element
NvSciSyncAttrList waiterAttr;
Application_GetEngineWaitAttrs(&waiterAttr);
// Indicate that the element will be used and provide the sync attrs
NvSciStreamBlockElementWaiterAttrSet(block, 1, waiterAttr);

// Inform NvSciStream that waiter attribute export is done
NvSciStreamBlockSetupStatusSet(block,
                               NvSciStreamSetup_WaiterAttrExport,
                               true);
```

Packets

Specifying packets

After calling `NvSciStreamBlockSetupStatusSet()` to indicate it is done exporting the element layout, the pool application can begin to define the packets. The functions to create packets and insert buffers into them remain the same as in 5.2. However, there is a new function which must be called to indicate a packet's specification is complete and send it to the rest of the stream. If the application previously waited for packet acceptance before sending the buffers or for buffer acceptance before sending the next buffer, it should no longer do so. The entire packet will now be sent at once.

```
NvSciError
NvSciStreamPoolPacketComplete(
    NvSciStreamBlock const pool,
    NvSciStreamPacket const handle);
```

As described below, the producer and consumer(s) will similarly accept or reject a packet and all its buffers at once. When the producer and all consumers have provided status for a given packet, the pool receives a single event, rather than separate events, for the packet and each buffer. To determine the status, there are two new queries. The first quickly checks whether the packet was accepted or rejected. If status has not yet been received for a packet, an error that indicates this will be returned. The second retrieves the status values sent by each endpoint, which can be used in the event the packet was rejected to learn more information.

Motivation for Change: Consolidating the events to send packets and receive status simplifies application organization.

```
typedef enum {
    …
    NvSciStreamEventType_PacketStatusProducer,
    NvSciStreamEventType_PacketStatusConsumer,
    NvSciStreamEventType_ElementStatusProducer,
    NvSciStreamEventType_ElementStatusConsumer,
    NvSciStreamEventType_PacketStatus,
    …
};


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);
```

After sending all packets, the application should indicate that packet setup is complete by calling the new `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_PacketExport`. After receiving the status for all packets, the application should call this function with a value of NvSciStreamSetup_PacketImport. An application may wait for status to be returned before indicating that packet export is finished, in case any of the packets was rejected and it wants to adjust something. This is left to the developers to decide.

Example

Old Code

```
NvSciBufObj buffer[][] = …
…
for (uint32_t p=0; p<packetCount; ++p) {
    NvSciStreamPacket packet;
    NvSciStreamPoolPacketCreate(pool, p+1, &packet);
    for (uint32_t b=0; b<bufferCount; ++b) {
        NvSciStreamPacketInsertBuffer(pool, packet, b, buffer[p][b]);
    }
}
bool packetFailure = false;
…
case NvSciStreamEventType_PacketStatusProducer:
    if (event.error != NvSciError_Success) {
        printf("Producer rejected packet %x with error %x\n",
                event.packetCookie, event.error);
        packetFailure = true;
    }
    break;
case NvSciStreamEventType_ElementStatusProducer:
    if (event.error != NvSciError_Success) {
        printf("Producer rejected buffer %d of packet %x "
                "with error %x\n",
                event.index, event.packetCookie, event.error);
        packetFailure = true;
    }
    break;
case NvSciStreamEventType_PacketStatusConsumer:
    if (event.error != NvSciError_Success) {
        printf("A consumer rejected packet %x with error %x\n",
                event.packetCookie, event.error);
        packetFailure = true;
    }
    break;
case NvSciStreamEventType_ElementStatusConsumer:
    if (event.error != NvSciError_Success) {
        printf("A consumer rejected buffer %d of packet %x "
                "with error %x\n",
                event.index, event.packetCookie, event.error);
        packetFailure = true;
    }
    break;
…
```

New Code

There are multiple ways the status handling could be organized. In particular, waiting for status could be interleaved with sending the packets instead. This is just one example that illustrates all the new functions.

```
NvSciBufObj buffer[][] = …
…
for (uint32_t p=0; p<packetCount; ++p) {
    NvSciStreamPacket packet;
    NvSciStreamPoolPacketCreate(pool, p+1, &packet);
    for (uint32_t b=0; b<bufferCount; ++b) {
        NvSciStreamPacketInsertBuffer(pool, packet, b, buffer[p][b]);
    }
    NvSciStreamPacketComplete(pool, packet);
}

NvSciStreamBlockSetupStatusSet(pool,
                               NvSciStreamSetup_PacketExport,
                               true);


bool packetFailure = false;
uint32_t packetsReady = 0;
…
case NvSciStreamEventType_PacketStatus:
    // Wait until status has arrived for all packets
    if (++packetsReady < packetCount) {
        break;
    }

    // Check each packet
    for (uint32_t p=0; p<packetCount; ++p) {
        // Check packet acceptance
        bool accept;
        NvSciStreamPoolPacketStatusAcceptGet(pool, packet[p], &accept);
        if (accept) {
            continue;
        }
        packetFailure = true;

        // On rejection, query and report details
        NvError status;
        NvSciStreamPoolPacketStatusValueGet(
            pool, packet[p], NvSciStreamBlockType_Producer, 0,
            &status);
        if (status != NvSciError_Success) {
            printf("Producer rejected packet %x with error %x\n",
```

```
                    packet[p], status);
        }
        for (uint32_t c=0; c<numConsumers; ++c) {
            NvSciStreamPoolPacketStatusValueGet(
                pool, packet[p], NvSciStreamBlockType_Consumer, c,
                &status);
            if (status != NvSciError_Success) {
                printf("Consumer %d rejected packet %x "
                        "with error %x\n",
                        c, packet[p], status);
            }
        }
    }

    // Inform NvSciStream that packet status import is done
    NvSciStreamBlockSetupStatusSet(pool,
                                    NvSciStreamSetup_PacketImport,
                                    true);
    break;
…
```

Receiving Packets

Rather than receiving separate events for creating a packet and each of its buffers, the producer and consumer(s) will receive a `NvSciStreamEventType_PacketCreate` event for each packet. They call a new function to dequeue the handle of the new packet and then another new function to obtain the buffers for any elements in the packet that they use.

```
typedef enum {
    …
    NvSciStreamEventType_PacketCreate,
    NvSciStreamEventType_PacketElement,
    …
};

NvSciError
NvSciStreamBlockPacketNewHandleGet(
    NvSciStreamBlock const block,
    NvSciStreamPacket* const handle);

NvSciError
NvSciStreamBlockPacketBufferGet(
    NvSciStreamBlock const block,
    NvSciStreamPacket const handle,
    uint32_t elemIndex),
    NvSciBufObj* const bufObj);
```

After checking whether it can map in all the buffers for a given packet, the endpoint should signal status back to the pool. Rather than sending separate status for the packet and each buffer, only a single function is called.

```
NvSciError
NvSciStreamBlockPacketAccept(
NvSciStreamBlockPacketStatusSet(
    NvSciStreamBlock const block,
    NvSciStreamPacket const handle,
    NvSciStreamCookie const cookie,
    NvSciError const status);

NvSciError
NvSciStreamBlockElementAccept(
    NvSciStreamBlock const block,
    NvSciStreamPacket const handle,
    uint32_t const index,
    NvSciError const err);
```

After the pool has indicated that it is finished exporting all the packets, the endpoints will receive a new `NvSciStreamEventType_PacketsComplete` event. They can finish up any setup related to packet resources and should then indicate they are done importing the packets by calling the `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_PacketImport`.

If a packet is deleted by the pool, when the producer or consumer receives the `NvSciStreamEventType_PacketDelete` event, it can determine the identify of the deleted packet by calling a function which retrieves the cookie of a packet pending deletion. After this function is called, the handle of the returned packet becomes invalid for subsequent function calls.

```
NvSciError
NvSciStreamBlockPacketOldCookieGet(
    NvSciStreamBlock const block,
    NvSciStreamCookie* const cookie);
```

Example

Old Code

```
…
case NvSciStreamEventType_PacketCreate:
    // Create new application-specific data structure for packet
    MyPacket* cookie = Application_CreateNewPacket(event.packetHandle);
    // Inform pool of success
    NvSciStreamBlockPacketAccept(
        block, cookie->packetHandle, cookie, NvSciError_Success);
    break;
```

```
case NvSciStreamEventType_PacketElement:
    // Retrieve application's data structure for the packet
    MyPacket* cookie = (MyPacket*)event.cookieHandle;
    // Map buffer into application
    Application_MapPacketBuffer(cookie, event.index, event.bufObj);
    // Inform pool of success
    NvSciStreamBlockElementAccept(
        block, cookie->packetHandle, event.index, NvSciError_Success);
    break;
…
```

New Code

```
…
case NvSciStreamEventType_PacketCreate:
    // Retrieve handle for packet pending creation
    NvSciStreamPacket packetHandle;
    NvSciStreamBlockPacketHandleGet(block, &packetHandle);
    // Create new application-specific data structure for packet
    MyPacket* cookie = Application_CreateNewPacket(packetHandle);
    // Retrieve all buffers and map into application
    //   Consumers can skip querying elements that they don't use
    for (uint32_t j=0; j<2; ++j) {
        NvSciBufObj bufObj;
        NvSciStreamBlockPacketBufferGet(
            block, packetHandle, j, &bufObj);
        Application_MapPacketBuffer(cookie, j, bufObj);
    }
    // Inform pool of success
    NvSciStreamBlockPacketStatusSet(
        block, packetHandle, cookie, NvSciError_Success);
    break;
case NvSciStreamEventType_PacketsComplete:
    // Note there is no corresponding operation in the old code because
    //   previously there was no way to inform the endpoints that the
    //   packet definition was done
    Application_FinalizePackets();
    NvSciStreamBlockSetupStatusSet(pool,
                                   NvSciStreamSetup_PacketImport,
                                   true);
    break;
…
```

Sync Objects

Specifying Sync Objects

After indicating their waiter sync attributes for all elements they support, by calling `NvSciStreamBlockElementWaiterAttrSet()`, the producer and consumer(s) must call the new `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_WaiterAttrExport`. This sends the sync attributes to the opposing endpoint(s), each receiving a single `NvSciStreamEventType_WaiterAttr` event.

The endpoints should query the waiter sync attributes for each element they support using a new function. (As in 5.2, if there are multiple consumers, their waiter sync attribute lists will be combined before they arrive at the producer, but now there will be a separate list for each element.) If NULL is received for any attribute list, it means one or more of the opposing endpoints do not support sync objects for that element, and the data should be written or read synchronously.

```
typedef enum {
    …
    NvSciStreamEventType_SyncAttr,
    NvSciStreamEventType_WaiterAttr,
    …
};

NvSciError
NvSciStreamBlockElementWaiterAttrGet(
    NvSciStreamBlock const block,
    uint32_t const elemIndex,
    NvSciSyncAttrList* const waitSyncAttrList);
```

For each element, the endpoints should combine and reconcile the received waiter attribute lists with their signaler attribute lists, and where appropriate, use them to create a sync object that they will signal. This occurs in 5.2, except there now one sync object per element. The function for sending global sync objects is replaced with a per-element function. It is no longer necessary to send the number of sync objects. If this function is not called for an element, the sync object is assumed to be NULL.

```
NvSciError
NvSciStreamBlockSyncObjCount(
    NvSciStreamBlock const block,
    uint32_t const count);

NvSciError
NvSciStreamBlockSyncObject(
NvSciStreamBlockElementSignalObjSet(
    NvSciStreamBlock const block,
    uint32_t const index,
    uint32_t const elemIndex,
```

```
    NvSciSyncObj const signalSyncObj);
```

After receiving all the sync attribute lists they care about, the endpoints should call the new `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_WaiterAttrImport`. After specifying all their sync objects, they should call it with a value of `NvSciStreamSetup_SignalObjExport`.

Example

Old Code

```
…
case NvSciStreamEventType_SyncAttr:
    NvSciSyncAttrList inputAttrs[2], combined, conflicts;
    // Obtain signaler attributes for engine
    Application_GetEngineSignalAttrs(&inputAttrs[0]);
    // Combine and reconcile with incoming waiter attributes
    inputAttrs[1] = event.syncAttrList;
    NvSciSyncAttrListReconcile(inputAttrs, 2, &combined, &conflicts);
    // Create new sync object
    NvSciSyncObj syncObj;
    NvSciSyncObjAlloc(combined, &syncObj);
    // Pass sync object to other end of stream
    NvSciStreamBlockSyncObjCount(block, 1);
    NvSciStreamBlockSyncObject(block, 0, syncObj);
…
```

New Code

```
case NvSciStreamEventType_WaiterAttr:
    NvSciSyncAttrList inputAttrs[2], combined, conflicts;
    // Retrieve waiter attribute for asynchronous element
    NvSciStreamBlockElementWaiterAttrGet(block, 1 &inputAttrs[1]);
    // Inform NvSciStream that sync attribute import is done
    NvSciStreamBlockSetupStatusSet(block,
                                   NvSciStreamSetup_WaiterAttrImport,
                                   true);
    // Obtain signaler attributes for engine
    Application_GetEngineSignalAttrs(&inputAttrs[0]);
    // Combine and reconcile
    NvSciSyncAttrListReconcile(inputAttrs, 2, &combined, &conflicts);
    // Create new sync object
    NvSciSyncObj syncObj;
    NvSciSyncObjAlloc(combined, &syncObj);
    // Associate sync object with asynchronous element
    NvSciStreamBlockElementSignalObjSet(block, 1, syncObj);
    // Inform NvSciStream that sync object export is done
```

```
    NvSciStreamBlockSetupStatusSet(block,
                                   NvSciStreamSetup_SignalObjExport,
                                   true);
    break;
…
```

Receiving Sync Objects

When the producer finishes specifying all its sync objects, the consumer(s) will receive a single `NvSciStreamEventType_SignalObj` event. Similarly, when all consumers have specified their sync objects, the producer will receive a single `NvSciStreamEventType_SignalObj` event.

The endpoints can then query these sync objects and map them into their engines. The producer will receive a separate sync object (or NULL) for each consumer.

```
typedef enum {
    …
    NvSciStreamEventType_SyncCount,
    NvSciStreamEventType_SyncDesc,
    NvSciStreamEventType_SignalObj,
    …
};

NvSciError
NvSciStreamBlockElementSignalObjGet(
    NvSciStreamBlock const block,
    uint32_t const queryBlockIndex,
    uint32_t const elemIndex,
    NvSciSyncObj* const signalSyncObj);
```

After receiving and mapping all the sync objects they care about, the endpoints should call the new `NvSciStreamBlockSetupStatusSet()` function with a value of `NvSciStreamSetup_SignalObjImport`.

Example

Old Code

```
…
case NvSciStreamEventType_SyncCount:
    Application_InitializeSyncList(event.count);
    break;
case NvSciStreamEventType_SyncDesc:
    Application_MapWaiterSync(event.index, event.syncObj);
    break;
…
```

New Consumer Code

```
…
case NvSciStreamEventType_SignalObj:
    Application_InitializeSyncList(1);
    NvSciSyncObj syncObj;
    NvSciStreamBlockElementSignalObjGet(block, 0, 1 syncObj);
    Application_MapWaiterSync(0, syncObj);
    break;
…
```

New Producer Code

```
…
case NvSciStreamEventType_SignalObj:
    Application_InitializeSyncList(numConsumers);
    for (uint32_t c=0; c<numConsumers; ++c) {
        NvSciSyncObj syncObj;
        NvSciStreamBlockElementSignalObjGet(block, c, 1 syncObj);
        Application_MapWaiterSync(c, syncObj);
    }
    break;
…
```

Phase Change

When the pool indicates it exported all the packets, and the endpoints indicate that they are done importing the packets and importing and exporting the sync objects, all blocks will receive a new 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.

```
typedef enum {
    …
    NvSciStreamEventType_SetupComplete,
    …
};
```

Example

Previously, there was no specific mechanism to signal this to the application. It would not know to transition until the first NvSciStreamEventType_PacketReady event arrived.

New Code

```
…
case NvSciStreamEventType_SetupComplete:
```

```
    Application_PhaseTransition();
    break;
…
```

Streaming Functions

The producer will continue to obtain packets to fill with `NvSciStreamProducerPacketGet()` and present them with `NvSciStreamProducerPacketPresent()`. Consumers will continue to obtain packets to process with `NvSciStreamConsumerPacketAcquire()` and return them for reuse with `NvSciStreamConsumerPacketRelease()`. However, the fence array parameters will be eliminated from these functions. Instead, the pre-fences associated with a packet can be queried one at a time after obtaining it, and post-fences can be specified one at a time before presenting it, using new functions.

```
NvSciError
NvSciStreamProducerPacketGet(
    NvSciStreamBlock const producer,
    NvSciStreamCookie *const cookie,
    NvSciSyncFence *const prefences);

NvSciError
NvSciStreamProducerPacketPresent(
    NvSciStreamBlock const producer,
    NvSciStreamPacket const handle,
    NvSciSyncFence const *const postfences);

NvSciError
NvSciStreamConsumerPacketAcquire(
    NvSciStreamBlock const consumer,
    NvSciStreamCookie *const cookie,
    NvSciSyncFence *const prefences);

NvSciError
NvSciStreamConsumerPacketRelease(
    NvSciStreamBlock const consumer,
    NvSciStreamPacket const handle,
    NvSciSyncFence const *const postfences);

NvSciError
NvSciStreamBlockPacketFenceSet(
    NvSciStreamBlock const block,
    NvSciStreamPacket const handle,
    uint32_t const elemIndex,
    NvSciSyncFence const *const postfence);

NvSciError
```

```
NvSciStreamBlockPacketFenceGet(
    NvSciStreamBlock const block,
    NvSciStreamPacket const handle,
    uint32_t const queryBlockIndex,
    uint32_t const elemIndex,
    NvSciSyncFence* const prefence);
```

Example

Only the producer is shown here. The consumer changes are analogous, except that there is only one incoming fence from the producer, so no loop is required.

Old Producer Code

```
…
case NvSciStreamEventType_PacketReady:
    MyPacket* cookie;
    NvSciSyncFence prefences[N], postfence;
    // Retrieve the packet to use
    NvSciStreamProducerPacketGet(
        producer, (NvSciStreamCookie*)&cookie, prefences);
    // Synchronize and generate the data
    for (uint32_t j=0; j<syncCount; ++j) {
        Application_EngineWaitForFence(prefences[j]);
    }
    Application_EngineGenerateData(cookie);
    Application_EngineSignalFence(&postfence);
    // Insert the finished packet in the stream
    NvSciStreamProducerPacketPresent(
        producer, cookie->packetHandle, &postfence);
    break;
…
```

New Producer Code

```
…
case NvSciStreamEventType_PacketReady:
    MyPacket* cookie;
    NvSciSyncFence fence;
    // Retrieve the packet to use
    NvSciStreamProducerPacketGet(
        producer, (NvSciStreamCookie*)&cookie);
    // Retrieve and wait for each consumer fence
    for (uint32_t j=0; j<numConsumer; ++j) {
        NvSciStreamBlockPacketFenceGet(
            producer, cookie->packetHandle, j, 1, &fence);
```

```
      Application_EngineWaitForFence(fence);
  }
  // Generate new data and signal fence
  Application_EngineGenerateData(cookie);
  Application_EngineSignalFence(&fence);
  // Update postfence for packet
  NvSciStreamBlockPacketFenceSet(
      Producer, cookie->packetHandle, 1, &fence);
  // Insert the finished packet into the stream
  NvSciStreamProducerPacketPresent(producer, cookie->packetHandle);
  break;
```

NvScilpc API Changes

Summary of the NvScilpc API Timeline

Release 6.0.2.0

- > Safe APIs in the Deprecated APIs section are not available
- > Migration is not required
- > Affected platform: Linux and QNX

Release 6.0.3.0

- > Both non-safe APIs and safe APIs in the Deprecated APIs section are available on standard and safety builds
- > APIs in the Modified APIs section are available on standard and safety builds
- > Migration to safe APIs is required
- > Affected platform: Linux and QNX

Release 6.0.4.0

- > Non-safe APIs are deprecated on standard and safety builds
- > Ensure code doesn't use non-safe APIs in the Deprecated APIs section
- > Affected platform: Linux and QNX

Release 6.0.5.0

- > Stakeholders must ensure the Safety build does not use non-safe APIs identified in 6.0.4.0
- > Affected platform: Linux and QNX

The NvSciIpc Library

The NvSciIpc library provides interfaces for any two 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
> The same process
> Different processes in the same VM
> Different VMs on the same SoC

Each of these different boundaries will be abstracted by a library providing unified communication (Read/Write) APIs to entities. The communication consists of two bi-directional send/receive queues.

Differences Between DRIVE OS 5.2 and 6.0

In 6.0.3.0,

> New safe APIs replace APIs from 5.2 to handle error detection and propagation paths more effectively. The move to safe APIs requires changes in the function signature, which breaks existing code unless the migration does not occur when the API is released.
> To support detection of write queue emptiness, a legacy API (`NvSciIpcGetEvent`) is updated ,which helps producers know when to fill in the buffer

In 6.0.4.0,

> Interrupt unmasking can occur even when the interrupt is not masked. New APIs enhance the interrupt unmasking flow so that no interrupt, which is not masked, is unmasked again. This new flow enhances performance
> Private pulse pool, introduced by QNX, prevents unprivileged processes from making interference-like DDoS attacks. New APIs support creating the channel with private pulse pool and inspecting which events are turned into rogue ones
> An event handler is supported in the event notifier of `NvSciEventService` and calls `WaitForXXX()` APIs when the event arrives
> Version checking APIs check library compatibility

Deprecated and Modified APIs

These APIs are replaced with safe versions, which strengthen error detection and the path to the returning error.

Deprecated:

```
void NvSciIpcCloseEndpoint(NvSciIpcEndpoint handle)
```

Safe version:

```
NvSciError NvSciIpcCloseEndpointSafe(NvSciIpcEndpoint handle, bool clear)
```

Improved safety points:

> The safe API checks bad parameters and returns an error if detected.

> The safe API has an option parameter to clear tx queue buffer for security.

Porting guideline:

```
NvSciError err;
bool clear;

clear = true; /* if tx buffer in queue wants to be cleared for security reason */
err = NvSciIpcCloseEndpointSafe(handle, clear);
/* add codes to handle err here */
```

Platform: QNX and Linux

Released: 6.0.3.0

Deprecated:

```
void NvSciIpcResetEndpoint(NvSciIpcEndpoint handle)
```

Safe version:

```
NvSciError NvSciIpcResetEndpointSafe(NvSciIpcEndpoint handle)
```

Improved safety points:

> The safe API checks for bad parameters and returns an error if detected.

Porting guideline:

```
NvSciError err;

err = NvSciIpcResetEndpointSafe(handle);
/* add codes to handle err here */
```

Platform: QNX and Linux

Released: 6.0.3.0

Deprecated:

```
NvSciError NvSciIpcRead(NvSciIpcEndpoint handle, void *buf, size_t size, int32_t *bytes)
```

Safe version:

```
NvSciError NvSciIpcReadSafe(NvSciIpcEndpoint handle, void *buf, uint32_t size, uint32_t
*bytes)
```

Improved safety points:

> The safe API limits the boundary of size and bytes parameter to unsigned integer 32bit

Porting guideline:

```
NvSciError err;
uint32_t size;
uint32_t bytes;

err = NvSciIpcReadSafe(handle, buf, size, bytes);
/* add codes to handle err here */
```

Platform: QNX and Linux

Released: 6.0.3.0

Deprecated:

```
NvSciError NvSciIpcWrite(NvSciIpcEndpoint handle, const void *buf, size_t size, int32_t
*bytes)
```

Safe version:

```
NvSciError NvSciIpcWriteSafe(NvSciIpcEndpoint handle, const void *buf, uint32_t size,
uint32_t *bytes)
```

Improved safety points:

The safe API limits the boundary of size and bytes parameter to unsigned integer 32bit.

Porting guideline:

```
NvSciError err;
uint32_t size;
uint32_t bytes;

err = NvSciIpcWriteSafe(handle, buf, size, bytes);
/* add codes to handle err here */
```

Platform: QNX and Linux

Released: 6.0.3.0

Deprecated:

```
NvSciError NvSciEventLoopServiceCreate(size_t maxEventLoops, NvSciEventLoopService**
newEventLoopService)
```

Safe version:

```
NvSciError NvSciEventLoopServiceCreateSafe(
        size_t maxEventLoops,
        void* config,
        NvSciEventLoopService** newEventLoopService);
```

Improved safety points:

> For QNX, it creates a private pulse pool that prevents DDOS attacks on the global pulse pool. The parameter config is passed to QNX API, ChannelCreatePulsePool(). The API depending on config sends semaphore event to a thread waiting the event when there is no room in pulse pool. Then the thread can do something to block suspicious event strom with the help of NvSciEventInspect()

> For Linux, the functionality is same as unsafe one

Porting guideline:

```
NvSciEventLoopService *eventLoopService;
sem_t *sem_noti;
NvSciError err;
/* Assume a thread handling there are no available pulses in the pool */
struct nto_channel_config config = {
    .num_pulses = 5, /* number of pulses queue */
    .rearm_threshold = 0, /* 0 fires once and never rearms the notification with
semaphore */
    .options = _NTO_CHO_CUSTOM_EVENT; /* semaphore event will be sent to a thread */
};

static void *priv_pulse_error_handler(void *arg)
{
    NvSciError err;

    while (true) {
        sem_wait(sem_noti);
        err = NvSciIpcInspectEventQnx(...) /* refer to the NvSciIpcInspectEventQnx
section */
        /* handle err */
    }
}

sem_noti = sem_open(SEM_ANON, O_ANON | O_CREAT, 0777, 0);
SIGEV_SEM_INIT(&config.event, sem_noti);

err = NvSciEventLoopServiceCreateSafe(1, &config, &eventLoopService);
/* add codes to handle err here */

/*
 * Create a thread waiting for sem_noti semaphore and when wakes up by the
 * semaphore, do something like calling NvSciEventInspect() to block suspicious
 * events which cause event storm
 */

phtread_create(&tid, NULL, priv_pulse_error_handle, NULL);
```

Defining num_pulses in nto_channel_config

> The process depends on the number of pulses that arrive simultaneously but are not processed, in the worst case scenario.

> If the estimated pulse number is 10, the recommendation is to have 12 with 20% margin

Defining rearm_threshold in nto_channel_config

> The process indicates the condition of the dropped-pulse notification with semaphore

> 0: Fires once and never rearms

> 1 through num_pulses fires and rearms when the pool utilization drops below the value of rearm_threshold

> Greater than num_pulses: Permanently armed (be careful not to overwhelm the system)

For additional information, see [nto_channel_config](#)

Platform: QNX and Linux

Released: 6.0.4.0

Deprecated:

```
NvSciError NvSciIpcSetQnxPulseParam(NvSciIpcEndpoint handle, int32_t coid, int16_t
pulsePriority, int16_t pulseCode, void *pulseValue)
```

Safe version:

```
NvSciError NvSciIpcSetQnxPulseParamSafe(NvSciIpcEndpoint handle,
    int32_t coid, int16_t pulsePriority, int16_t pulseCode);
```

Improved safety points:

The safe version does not support an application cookie input parameter with `pulseValue` to reroute unmasking interrupt.

Porting guideline:

Refer to the porting guidelines in `MsgReceivePulse_r()`

Platform: QNX

Released: 6.0.4.0

Deprecated:

```
NvSciError NvSciIpcGetEvent(NvSciIpcEndpoint handle, uint32_t *events)
```

Safe version:

```
NvSciError NvSciIpcGetEventSafe(NvSciIpcEndpoint handle, uint32_t *events)
```

Improved safety points:

> It unmasks interrupt with the correct procedure from QNX manual

> It enhances Inter-VM throughput by removing unnecessary QNX kernel calls

> It supports QNX OS 7.2 muon kernel

> For Linux, the functionality remains the same

Porting guideline:

Refer to the porting guidelines in `MsgReceivePulse_r()`

Platform: QNX

Released: 6.0.4.0

Deprecated:

```
int MsgReceivePulse_r(int chid, void * pulse, size_t bytes, struct _msg_info * info)
```

Safe version:

```
int32_t NvSciIpcWaitEventQnx(int32_t chid, int64_t microseconds, uint32_t bytes, void *pulse)
```

Improved safety points:

> It unmasks interrupt with the correct procedure from QNX manual

> It supports timeout that is not blocked permanently

> This update is required only when `MsgReceivePulse_r()` is used to block IVC notification in D5.2

Porting guideline:

```
#define APP_PULSE_CODE 10
NvSciIpcEndpoint ipcEndpoint;
int32_t coid;
int16_t priority = SIGEV_PULSE_PRIO_INHERIT;
int16_t code = APP_PULSE_CODE; /* application-defined code value */
int64_t timeout = 10000; /* 10 msec */
NvSciError err;

/*
 * Create channel and open endpoint
 * coid and ipcEndpoint are initialized properly
 */

err = NvSciIpcSetQnxPulseParamSafe(ipcEndpoint, coid, priority, code);
/* add codes to handle err here */

/*
 * Get endpoint information and perform reset
 */

while (1) {
      err = NvSciIpcGetEventSafe(ipcEndpoint, &event);
      if (err != NvSciError_Success) {
```

```
            goto fail;
      }

      if (event & NV_SCI_IPC_EVENT_WRITE) {
      /* perform write operation */
      } else if (event & NV_SCI_IPC_EVENT_READ) {
      /* perform read operation */
      } else {
            err = NvSciIpcWaitEventQnx(chid, timeout, sizeof(pulse), &pulse);
            /* add codes to handle err here */
       }
}
```

Platform: QNX

Released: 6.0.4.0

Modified API:

```
NvSciError NvSciIpcGetEvent(NvSciIpcEndpoint handle, uint32_t *events)
```

The API adds one new event, which specifies that the remote endpoint receives all data that the local endpoint sends.

New event:

```
NV_SCI_IPC_EVENT_WRITE_EMPTY 0x10U  ; write fifo is empty
```

Porting guideline:

```
NvSciError err;
Uint32_t event;

err = NvSciIpcGetEvent(handle, &event);
/* add codes to handle err here */

If (event & NV_SCI_IPC_EVENT_WRITE_EMPTY) {
    /* write buffer in queue becomes empty now it is time to fill in the buffer   or
close connection  */
}
```

> **Note:** This API is deprecated in 6.0.4.0. `NvSciIpcGetEventSafe()` supports the same event: `NV_SCI_IPC_EVENT_WRITE_EMPTY`

New API:

```
NvSciError NvSciEventService::(*WaitForMultipleEventsExt)(NvSciEventService
*eventService, NvSciEventNotifier* const * eventNotifierArray, size_t eventNotifierCount,
int64_t microseconds, bool* newEventArray)
```

The new API is similar to `NvSciEventLoopService::WaitForMultipleEvents()` but functions differently, depending on microseconds when `eventNotifierArray` is NULL.

1. `microseconds > 0`
   Callbacks(event handler) continue to be served until the timeout occurs. If a callback is longer than the timeout, other callbacks associated with events, which arrive before timeout, are still served after timeout.

2. `microseconds = -1 (NV_SCI_EVENT_INFINITE_WAIT)`
   Callbacks continue to be served and this API never returns.

Porting guideline:

Refer to `NvSciEventLoopService::WaitForMultipleEvents()`

Platform: QNX and Linux

Released: 6.0.3.0 in Linux and 6.0.4.0 in QNX

New API:

```
NvSciError NvSciEventNotifier::(*SetHandler)(NvSciEventNotifier* thisEventNotifier, void
(*callback)(void* cookie), void* cookie, uint32_t priority)
```

It sets an event handler which is called in these functions when the pulse event associated with the notifier is arrived.

> NvSciEventLoopService::WaitForEvent() or

> NvSciEventLoopService::WaitForMultipleEvents() or

> NvSciEventLoopService::WaitForMultipleEventsExt()

Porting guideline:

```
NvSciIpcEndpoint ipcEndpoint;
NvSciEventNotifier *eventNotifier;
NvSciError err;
void * cookie;

void eventHandler(void *cookie)
{
    /* do something when it is called by the event */
}

/*
  * Open endpoint with EventService
  */

err = NvSciIpcGetEventNotifier(ipcEndpoint, &eventNotifier);
/* add codes to handle err here */

err = eventNotifier->SetHandler(eventNotifier, eventHandler, cookie, 0);
/* add codes to handle err here */
```

Platform: QNX and Linux

Released: 6.0.3.0 in Linux and 6.0.4.0 in QNX

New API:

```
NvSciError NvSciIpcInspectEventQnx(int32_t chid, uint32_t numPulses, uint32_t epCount,
NvSciIpcEndpoint** epHandleArray)
It inspects events in queue, makes a decision on a rogue event and unregisters the event.
```

Porting guideline:

```
#define MAX_ENDPOINT /* It must be greater than all opened endpoint count in application
process */
int32_t chid;
uint32_t numPulses; /* A threshold to unregister event */
uint32_t epCount = MAX_ENDPOINT;
NvSciIpcEndpoint epHandleArray[MAX_ENDPOINT];
NvSciIpcEndpoint *ptr = epHandleArray;
NvSciError err;

/*
  * Create channel and open endpoint
  * chid is initialized properly
  */

/* In a thread which wakes up by event indicating private pulse pool is full */
err = NvSciIpcInspectEventQnx(chid, numPulses, epCount, &epHandleArray)
/* add codes to handle err here */
```

Platform: QNX

Released: 6.0.4.0

New API:

```
NvSciError NvSciEventInspect(NvSciEventService *thisEventSerivce, uint32_t numEvents,
uint32_t eventNotifierCount, NvSciEventNotifier** eventNotifierArray)
```

The API inspects events in queue associated with `EventService`, makes a decision on a rogue event, and unregisters the event.

Porting guideline:

```
#define MAX_NOTIFIER /* It must be greater than all created notifier in application
process */
NvSciEventLoopService *eventLoopService;
uint32_t numEvents; /* A threshold to unregister event */
uint32_t notiCount = MAX_NOTIFIER;
NvSciEventNotifier *notiArray[MAX_NOTIFIER];
NvSciError err;

/*
  * Create EventLoopService and Notifiers
```

```
 * eventLoopService is initialized properly
 */
```

```
/* In a thread which wakes up by event indicating private pulse pool is full */
err = NvSciEventInspect(&eventLoopServiceP->EventService, numEvents, notiCount,
&notiArray[0])
/* add codes to handle err here */
```

Platform: QNX

Released: 6.0.4.0

New API:

```
NvSciError NvSciIpcCheckVersionCompatibility(uint32_t majorVer, uint32_t
minorVer,   bool* isCompatible)
```

The API indicates if the loaded NvSciIpc library is compatible with the given version.

Porting guideline:

```
bool compatible;
NvSciError err;

err = NvSciIpcCheckVersionCompatibility(NvSciIpcMajorVersion,
    NvSciIpcMinorVersion, &compatible);
if ((err != NvSciError_Success) || (compatible != true)) {
   /* there is an error or the version is not compatible */
}
```

Platform: QNX and Linux

Released: 6.0.4.0

New API:

```
NvSciError NvSciEventCheckVersionCompatibility(uint32_t majorVer, uint32_t
minorVer,   bool* isCompatible)
```

The API indicates if the loaded `NvSciEventService` library is compatible with the given version.

Porting guideline:

```
bool compatible;
NvSciError err;

err = NvSciEventCheckVersionCompatibility(NvSciEventMajorVersion,
    NvSciEventMinorVersion, &compatible);
if ((retval != NvSciError_Success) || (compatible != true)) {
   /* there is an error or the version is not compatible */
}
```

PKCS#11 API Changes

Table 37.   Migrating PKCS#11 Library from 5.2 to 6.0

The following table describes PKCS#11 Library changes from Drive OS 5.2 to 6.0

Header File and API	Change Description
Update from PKCS11-Base-v2.40 to PKCS11-Base-v3.0 headers	
Header File: nvpkcs11_future.h nvpkcs11f_future.h	Removed contents but retained files
Header File: pkcs11f.h **API:** ADDED C_GetInterfaceList C_GetInterface C_LoginUser C_SessionCancel C_MessageEncryptInit C_EncryptMessage C_EncryptMessageBegin C_EncryptMessageNext C_MessageEncryptFinal C_MessageDecryptInit C_DecryptMessage C_DecryptMessageBegin C_DecryptMessageNext C_MessageDecryptFinal C_SignMessageBegin C_SignMessageNext C_VerifyMessageBegin C_VerifyMessageNext	APIs added as a result of updating from PKCS11-Base-v2.40 to PKCS11-Base-v3.0 header
Header File: pkcs11t.h	Updated from PKCS11-Base-v2.40 to PKCS11-Base-v3.0 header
Header File: nvpkcs11.h	Moved version 3.0 message based APIs from NV_CK_FUNCTION_LIST as these are now included in CK_FUNCTION_LIST_3_0. PKCS#11 library supports 3 interfaces:

Header File and API	Change Description

API:
REMOVED
 	
    `C_NvGetFunctionList`
 	
MOVED
 	
    `C_MessageSignInit`
    `C_SignMessage`
    `C_MessageSignFinal`
    `C_MessageVerifyInit`
    `C_VerifyMessage`
    `C_MessageVerifyFinal`
	"PKCS 11": this interface name represents 2 interfaces, one which is associated with the Oasis standards version 3.0 `CK_FUNCTION_LIST_3_0` structure and the other with the Oasis standards version 2.40 `CK_FUNCTION_LIST` structure.
	
	"Vendor NVIDIA": this interface name is associated with `NV_CK_FUNCTION_LIST` structure that contains NVIDIA extension APIs.

Header File:
 	
    `nvpkcs11.h`
 	
API:
REMOVED
 	
    `C_SignBatchMessage`
    `C_VerifyBatchMessage`
	Removed batch message extension APIs

PKCS#11 public APIs that changed in terms of signatures and parameters	
`C_UnwrapKey`	Changed supported unwrap mechanism from `CKM_AES_CCM` to `CKM_AES_GCM`. Changes within key metadata storage layout and content necessitate all keys prepared offline with PKCS#11 Object Generation Tool to be revisited using the tool aligned with the DRIVE OS release in use. `CKA_TOKEN` determines if a key is a Token key when unwrapped within a R/W session.
`C_DeriveKey`	Changed fused base key `CKA_ID` from `NV_OEM_KEK2`. Refer to PDK Documentation aligned with the DRIVE OS release in use. Fused base keys are `GENERIC_SECRET` type requiring the prf passed into `CKM_SP800_108_COUNTER_KDF` to be `SHA256_HMAC`.

Header File and API	Change Description
	Template attribute `CKA_KEY_TYPE` mandatory for the derive operation. 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_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. `CKM_SP800_108_COUNTER_KDF` Input Parameters: The byte 0x00 is not allowed within Label or Context of the PRF input data fields
`C_FindObjectsInit`	Template attribute `CKA_CLASS` is mandatory.
All APIs associated with a template	`CKA_CHECK_VALUE` is not supported `CKA_UNIQUE_ID` is not supported

Release 6.0.5.0	
Header File: nvpkcs11.h **API:** ADDED `C_NVIDIA_CommitTokenObjects`	NVIDIA extension API added in support of committing persistent object changes to secure storage
Header File: nvpkcs11.h **API:** RENAMED	All NVIDIA extension APIs shall start "C_NVIDIA_..." `C_EncryptGetIV` is still accepted for backwards compatibility.

| ```
    C_EncryptGetIV
TO
    C_NVIDIA_EncryptGetIV
``` | |
Release 6.0.8	
API: `C_FindObjectsInit`	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)
Release 6.0.8.1	

Header File:
 	
    `nvpkcs11.h`
 	
API:
RENAMED
 	
    `CKM_AES_GCM`
TO
 	
    `CKM_NVIDIA_AES_GCM_KEY_UNWRAP`
	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.

Release 6.0.9.1	

Header File:
 	
    `nvpkcs11_public_defs.h`
 	
API:
REPLACED
 	
    `ecdsaPoint_t`
    `eddsaPoint_t`
WITH
 	
    `uncompressed32BytePoint_t`
    `compressed32BytePoint_t`
	`ecdsaPoint_t` and `eddsaPoint_t` are still accepted as legacy alias types for backwards compatibility.

Release 6.0.9.1	

Header File:
 	
    `nvpkcs11_public_defs.h`
 	
APIs:
DEPRECATED
 	
    `NVPKCS11_SHA_MAX_DATA_SIZE 256UL *`
    `0x100000UL`
 	
    `NVPKCS11_AES_BUFFER_LIMIT 1U *`
    `0x100000UL`
	Since Release 6.0.8, 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*.
	
	Applicable max buffer sizes should be determined from knowledge of your device tree channel settings.

Header File:
 	
    `nvpkcs11_public_defs.h`
 	
APIs:
DEPRECATED
 	
    `NVPKCS11_ECDSA_256_KEY_SIZE`
    `NVPKCS11_EDDSA_256_KEY_SIZE`
 	
REPLACED WITH
 	
    `NVPKCS11_EC_256_POINT_SIZE`
	Macro definitions based on NVIDIA hardware and software capabilities.

PKCS#11–Implementation Details

Release 6.0.5.0

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--CCPLEX (the largest set of cryptographic hardware support), TSEC (supports AES CMAC sign and verify exclusively), and FSI (key management only, no crypto operations supported). These must be represented in different PKCS#11 tokens, because they represent different hardware.

There is a requirement for object access control for CCPLEX. This allows different applications to use the same CCPLEX hardware, but with access to different sets of objects. For example, an application can processes sensor data with a 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 be able to 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 also 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. Each configuration of hardware and access control has 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 from stalling as the commit happens)
> Either reboot or go through an SC7 cycle (calling `C_Finalize()` before suspending and `C_Initialize()` after resuming)

The changes from a single token to multiple tokens are introduced in stages. The 6.0.5.0 release supports four tokens.

Refer to the documentation accompanying Release 6.0.5.0 for further details regarding which token to use; to view an example showing how to perform secure storage updates, and for more information about isolation between safety and dynamic tokens.

Release 6.0.6.0

Accessing and Choosing a Token

The existing client application code must be updated to choose a token explicitly, and to use the new GIDs and custom abilities.

Refer to the documentation accompanying release 6.0.6 for further details.

Board Support Packages (BSP) API Changes

Table 38.   QNX BSP Driver

Device	Change Description
NvGPIO	The `NvGpioOutputLoopbackDiag()` API is new and supports Output loopback diagnostic testing for a safety-critical output GPIO pin. The `NvGpio_OutputLoopback Err` return value is new and indicates failure of the Output loopback diagnostic test. Customers can perform the Output loopback diagnostic test for a safety-critical output GPIO pin using the `NvGpioOutputLoopbackDiag()` API. Note: Check the return value from the NvGpioOutputLoopbackDiag() API.
I2C Driver	`DCMD_I2C_DRIVER_INFO` is deprecated and should be removed Operation mode of `DCMD_I2C_BUS_RESET` changed from `INIT/DEINIT` to `INIT/RUNTIME/DEINT`. `DCMD_I2C_BUS_RESET` is available anytime The return values of QNXBSP I2C API changed. Customers should be careful when checking for the return value from the QNXBSP I2C driver i2c.h is deprecated and replaced with nvi2c_devctl.h. Custmers should replace i2c.h with nvi2c_devctl.h
NvThermmon	The API to read zone temperature has changed. A previous version of the NvThermmon API was deprecated in NVIDIA DRIVE OS 6.0.4.0, and the change was communicated in the 6.0.3.0 release notes. **Old API:** `NvThermmonErrCode NvThermmonGetZoneTemp(const NvThermmonHandle Handle, int32_t * const TempVal);` **New API:**

Device	Change Description

	```NvThermmonErrCode NvThermmonGetZoneTemp(const NvThermmonHandle Handle, struct zone_temp_t * const zone_temp)```
	
	The new API returns a structure of type `zone_temp_t`, which has two consecutive temperature read values along with the corresponding timestamp.
	
	Porting Guideline:
	
	```NvThermmonHandle Handle = NULL;\nchar * zone_name = "tj-therm";\nstruct zone_temp_t zone_temp;\n\nNvThermmonOpen(zone_name, &Handle);\nNvThermmonGetZoneTemp(Handle, &zone_temp);\nprintf("Temp for zone %s:\ntime_stamp\t temp_val\n", zone_name);\nprintf("0x%lx\t %d\n", zone_temp.ts_1, zone_temp.temp_1);\nprintf("0x%lx\t %d\n", zone_temp.ts_2, zone_temp.temp_2);```
	
	Both temperature readings are consecutive and there is not much difference between the two readings. For the latest reading, the client should use the zone_temp.temp_2 value.

DRIVE Update Changes

The following table describes Drive Update changes from Drive 5.1 to 6.0

Table 39.   Drive Update Changes from 5.1 to 6.0

Header File and API	Change Description

Header File:
 	
   nvdriveupdate.h
 	
API:
 	
   lcapi
	> LCAPI has been deprecated since 5.2 and is removed in 6.0. This applies to the following APIs:
	• NvStartDeployPackage
	• NvCancelDeployPackage
	• NvGetDeployPackageProgress
	• NvSwitchBootChain
	• NvQueryBootChain
	• NvReadPartitionContent
	• NvQueryIsPartitionPresent

Header File and API	Change Description
	• NvErasePartitionContent
	• NvQueryIsPartitionErased
	• NvSetRateGate
	> Check PDK documentation for migration from LCAPI to new APIs
	> The following `typedefs` were removed:
	• `NvPkgType`
	• `NvBootChain`
	• `NV_DU_RATEGATE_CB`

The following table describes Drive Update changes from Drive OS 5.2 to 6.0.

Table 40.   Drive Update Changes from 5.2 to 6.0

Header File and API	Change Description

Header File:
 	
`dulink.h`
 	
API:
 	
`dulink`
	> The pBuf parameter is changing from `void*` to `void const*`
	> Following typedefs were removed
	• `PDULINK_ACL_ENTRY`
	• `PDULINK_ATTR`

Header File:
 	
`dutransport.h`
 	
API:
 	
`dutransport`
	> The following `typedefs` will be removed
	• `PDUTR_SEC_PARAM`
	• `PDUTR_IVC_PARAM`
	• `PDUTR_SHMITC_PARAM`
	• `DUTR_SHMIPC_PARAM, PDUTR_SHMIPC_PARAM`
	• `PDUTR_TCP_IF_PARAM`
	• `PDUTR_TCP_PARAM`
	• `PDUTR_TR_PARAM`
	> The following enums were removed
	• `DUTR_TR_TYPE_SHM_ITC`

Metadata	A new optional parameter updates chain C or chain D. There's no impact on the current update package if you only plan to use chain A or chain B

NVIDIA DRIVE OS VM State Management (NvDVMS)

Note:	For changes to NvDVMS, refer to the NVIDIA DRIVE OS releases after release 6.0.4.0.
	For additional information regarding NvDVMS, refer to the NVIDIA DRIVE OS PDK and SDK developer guides.

Table 41.   API and Interface Changes for for 6.0.9.2

API/Interface	Affected	Action	Impact	Impacted Release
Access to NvDVMS resource manager device nodes is restricted to owner and group only. Earlier it was enabled for all.	All applications having direct/indirect dependency on NvDVMS.	DRIVE OS user needs to update the launch command of applications dependent on NvDVMS, to include NvDVMS UID (2100) in the SGID list	Launch of all applications having direct/indirect dependency on NvDVMS will fail.	6.0.9.2

TensorRT API Changes

Note:	For changes to Tensor RT after TensorRT version 8.4 and DRIVE OS version 6.0.5, refer to the TensorRT Release Notes.

Table 42   API Changes

The following table describes changes for 6.0.1, 6.0.2, and 6.0.3.

Interface	Affected	Action	Impact	Impacted Release
`ILogger::log`	Logging interface	Thread safety is now documented as being required for	Custom logging implementations of ILogger::log provided by applications that	6.0.3

Interface	Affected	Action	Impact	Impacted Release
		implementations of `ILogger::log`. Previous documentation did not require this.	are not thread safe must be rewritten to be thread safe.	
`getPluginRegistry`	Safe runtime	Added `getSafePluginRegistry` for safe runtime.	Impacts user code.	6.0.1
`REGISTER_TENSORRT_PLUGIN`	Safe runtime	Renamed to `REGISTER_SAFE_TENSORRT_PLUGIN`.	Impacts user code.	6.0.1
`IPluginChecker::getPluginType`	Standard and safe runtimes	Renamed to `getPluginName` for consistency.	Impacts user code.	6.0.1
`IPluginRegistry::deregisterCreator`	Standard and safe runtimes	New method for deregistering plugins.	No impact to existing code.	6.0.1
`IPluginChecker`	Consistency checker	Updated to inherit from `IPluginCreator`.	Impacts user code	6.0.1
`IPluginCheckerRegistry::registerPlugin`	Consistency checker	Use `IPluginRegistry::registerCreator`.	Impacts user code	6.0.1
`IPluginCheckerRegistry::getPluginChecker`	Consistency checker	Use `IPluginRegistry::getPluginCreator`.	Impacts user code	6.0.1
`IPluginCheckerRegistry`	Consistency checker	Removed.	Impacts user code	6.0.1

Interface	Affected	Action	Impact	Impacted Release
`NvInferRuntimeSelect.h`	Standard and safe runtimes	Removed.	Impacts user code	6.0.1
`NvInferConsistency.h`	Consistency checker	Updated to use pointer-to-implementation.	No impact to user code	6.0.1
Proxy runtime split into `libnvinfer_safe.so`	Proxy runtime	Proxy runtime split from `libnvinfer.so` to separate library.	Impacts user code (linking)	6.0.1
`IBuilderConfig::getMemoryPoolLimit` `IBuilderConfig::setMemoryPoolLimit`	TensorRT builder	New API	No impact to user code	6.0.2
`ICaffeParser::parseBuffers`	Caffe parser	Type updated from char* to uint8_t*	No impact to user code	6.0.2

Appendix: Additional Resources

For information about NVIDIA products and resources, refer to the NVIDIA Documentation Center and the NVIDIA DeveloperZone.