nvbuf_utils to NvUtils Migration Guide

Jetson Linux

Application Note

DA_10878-001 | March 2022

Document History

DA_10878-001

Version	Date	Authors	Description of Change
1.0	March 25, 2022	Amit Pandya	Initial release, r34.0.1 EA.

Table of Contents

Introduction

With NVIDIA® Jetson™ Linux release r34.0.1, Jetson Linux Multimedia introduces NvUtils, a new API for multimedia buffer management and for buffer transformation, including composition and blending. This user guide will facilitate your migration from the old API to the new one.

Added Features

Added features in the NvUtils API include:

- A new buffer allocation function with batched buffer allocation support
- A new buffer allocation function with CUDA memory allocation support
- Video transformation support using CUDA and the VIC hardware engine

Public Header Files and Libraries

iThe NvUtils API replaces nvbuf_utils, an older Jetson Linux API. For your convenience in migrating software from nvbuf_utils to NvUtils, this document describes both APIs' headers.

nvbuf_utils is declared by the C header file `nvbuf_utils.h`, and implemented in the shared library `libnvbuf_utils.so`. The latest API reference is Jetson Linux 32.7.1 Jetson Linux API Reference, included in the JetPack 4.6 distribution.

NvUtils is declared by the C header files `nvbufsurface.h` and `nvbufsurftransform.h`, and is implemented in the shared library `libnvbuf_utils.so`. The latest API reference is Jetson Linux API Reference for release 34.0.1, included in the JetPack 5.0 distribution.

Both libraries are included in release 34.0.1 of Jetson Linux BSP.

Ported Enum Definitions

This section describes enum definitions ported from nvbuf_utils to NvUtils.

nvbufsurface.h enums

This section describes ported enum definitions in `nvbufsurface.h`, concerning buffer allocation.

Buffer Payload Types	
NvBufferPayLoadType (nvbuf_utils.h)	**NvBufSurfaceMemType (nvbufsurface.h)**
	`NVBUF_MEM_DEFAULT` Specifies the default memory type: • `NVBUF_MEM_CUDA_DEVICE` for dGPU • `NVBUF_MEM_SURFACE_ARRAY` for Jetson Use `NVBUF_MEM_DEFAULT` to allocate whichever type of memory is appropriate for the platform.
	`NVBUF_MEM_CUDA_PINNED` Specifies CUDA host memory type.
	`NVBUF_MEM_CUDA_DEVICE` Specifies CUDA device memory type.
	`NVBUF_MEM_CUDA_UNIFIED` Specifies CUDA unified memory type.
`NvBufferPayload_SurfArray` Buffer payload with hardware memory handle for a set of planes.	`NVBUF_MEM_SURFACE_ARRAY` Specifies NVRM Surface Array type. Valid only for Jetson.
`NvBufferPayload_MemHandle` Buffer payload with hardware memory handle for a specific memory size.	`NVBUF_MEM_HANDLE` Specifies NVRM handle type. Valid only for Jetson.
	`NVBUF_MEM_SYSTEM` Specifies memory allocated by `malloc()`,

Video Plane Display Scan Formats	
NvBufferDisplayScanFormat (nvbuf_utils.h)	**NvBufSurfaceDisplayScanFormat (nvbufsurface.h)**
`NvBufferDisplayScanFormat_Progressive` Progressive scan formats.	`NVBUF_DISPLAYSCANFORMAT_PROGRESSIVE` Progressive scan formats.
`NvBufferDisplayScanFormat_Interlaced` Interlaced scan formats.	`NVBUF_DISPLAYSCANFORMAT_INTERLACED` Interlaced scan formats.

Video Plane Layout Formats	
NvBufferLayout (nvbuf_utils.h)	**NvBufSurfaceLayout (nvbufsurface.h)**
`NvBufferLayout_Pitch` Pitch Layout.	`NVBUF_LAYOUT_PITCH` Specifies pitch layout.
`NvBufferLayout_BlockLinear` BlockLinear Layout.	`NVBUF_LAYOUT_BLOCK_LINEAR` Specifies block linear layout.

Buffer Memory Access Flags	
NvBufferMemFlags (nvbuf_utils.h)	**NvBufSurfaceMemMapFlags (nvbufsurface.h)**
`NvBufferMem_Read` Memory read.	`NVBUF_MAP_READ` Specifies mapping type read.
`NvBufferMem_Write` Memory write.	`NVBUF_MAP_WRITE` Specifies mapping type write.
`NvBufferMem_Read_Write` Memory read & write.	`NVBUF_MAP_READ_WRITE` Specifies mapping type read/write.

Client Memory Allocation Tag	
NvBufferTag (nvbuf_utils.h)	**NvBufSurfaceTag (nvbufsurface.h)**
`NvBufferTag_NONE` = 0x0 Tag for no allocation.	`NvBufSurfaceTag_NONE` = 0x0 Tag for no allocation.
`NvBufferTag_CAMERA` = 0x200 Tag for Camera.	`NvBufSurfaceTag_CAMERA` = 0x200 Tag for camera.
`NvBufferTag_JPEG` = 0x1500 Tag for JPEG encoder/decoder.	`NvBufSurfaceTag_JPEG` = 0x1500 Tag for JPEG encoder/decoder.

NvBufferTag (nvbuf_utils.h)	NvBufSurfaceTag (nvbufsurface.h)
NvBufferTag_PROTECTED = 0x1504 Tag for VPR buffers.	NvBufSurfaceTag_PROTECTED = 0x1504 Tag for VPR buffers.
NvBufferTag_VIDEO_ENC = 0x1200 Tag for video encoder.	NvBufSurfaceTag_VIDEO_ENC = 0x1200 Tag for video encoder.
NvBufferTag_VIDEO_DEC = 0x1400 Tag for video decoder	NvBufSurfaceTag_VIDEO_DEC = 0x1400 Tag for video decoder.
NvBufferTag_VIDEO_CONVERT = 0xf01 Tag for video transform/composite.	NvBufSurfaceTag_VIDEO_CONVERT = 0xf01 Tag for video transform/composite/blend.

nvbufsurftransform.h enums

This section describes ported enum definitions in nvbufsurftransform.h, concerning buffer transformations.

Video Flip Methods	
NvBufferTransform_Flip (nvbuf_utils.h)	**NvBufSurfTransform_Flip (nvbufsurftransform.h)**
NvBufferTransform_None Video flip none.	NvBufSurfTransform_None Specifies no video flip.
NvBufferTransform_Rotate90 Video flip rotates 90-degree counter-clockwise.	NvBufSurfTransform_Rotate90 Specifies rotating 90 degrees clockwise.
NvBufferTransform_Rotate180 Video flip rotates 180 degrees.	NvBufSurfTransform_Rotate180 Specifies rotating 180 degrees clockwise.
NvBufferTransform_Rotate270 Video flip rotates 270 degrees counterclockwise.	NvBufSurfTransform_Rotate270 Specifies rotating 270 degrees clockwise.
NvBufferTransform_FlipX Video flip with respect to X-axis.	NvBufSurfTransform_FlipX Specifies video flip with respect to the X-axis.
NvBufferTransform_FlipY Video flip with respect to Y-axis.	NvBufSurfTransform_FlipY Specifies video flip with respect to the Y-axis.
NvBufferTransform_Transpose Video flip transpose.	NvBufSurfTransform_Transpose Specifies video flip transpose.
NvBufferTransform_InvTranspose Video flip inverse transpose.	NvBufSurfTransform_InvTranspose Specifies video flip inverse transpose.

The “Video Transform Flags” represent valid transform operations.

Video Transform Flags	
NvBufferTransform_Flag (nvbuf_utils.h)	**NvBufSurfTransform_Transform_Flag (nvbufsurftransform.h)**
NVBUFFER_TRANSFORM_CROP_SRC = 1 Transform flag to crop source rectangle.	NVBUFSURF_TRANSFORM_CROP_SRC Specifies a transform to crop the source rectangle.
NVBUFFER_TRANSFORM_CROP_DST = 1 << 1 Transform flag to crop destination rectangle.	NVBUFSURF_TRANSFORM_CROP_DST Specifies a transform to crop the destination rectangle.
NVBUFFER_TRANSFORM_FILTER = 1 << 2 Transform flag to set filter type.	NVBUFSURF_TRANSFORM_FILTER Specifies a transform to set the filter type.
NVBUFFER_TRANSFORM_FLIP = 1 << 3 Transform flag to set flip method.	NVBUFSURF_TRANSFORM_FLIP Specifies a transform to set the flip method.
	NVBUFSURF_TRANSFORM_NORMALIZE Specifies a transform to normalize output.

Video Filter Types	
NvBufferTransform_Filter (nvbuf_utils.h)	**NvBufSurfTransform_Inter (nvbufsurftransform.h)**
NvBufferTransform_Filter_Nearest Transform filter nearest.	NvBufSurfTransformInter_Nearest Specifies Nearest Interpolation Method interpolation.
NvBufferTransform_Filter_Bilinear Transform filter bilinear.	NvBufSurfTransformInter_Bilinear Specifies Bilinear Interpolation Method interpolation.
NvBufferTransform_Filter_5_Tap Transform filter 5 tap.	NvBufSurfTransformInter_Algo1 Specifies GPU-Cubic, VIC-5 Tap interpolation.
NvBufferTransform_Filter_10_Tap Transform filter 10 tap.	NvBufSurfTransformInter_Algo2 Specifies GPU-Super, VIC-10 Tap interpolation.
NvBufferTransform_Filter_Smart Transform filter smart.	NvBufSurfTransformInter_Algo3 Specifies GPU-Lanzos, VIC-Smart interpolation.
NvBufferTransform_Filter_Nicest Transform filter nicest.	NvBufSurfTransformInter_Algo4 Specifies GPU-Ignored, VIC-Nicest interpolation.
	NvBufSurfTransformInter_Default Specifies GPU-Nearest, VIC-Nearest interpolation.

The Video Composite Flags represent valid composition or blending operations.

Video Composite Flags	
NvBufferComposite_Flag (`nvbuf_utils.h`)	**NvBufSurfTransform_Composite_Flag** (`nvbufsurftransform.h`)
`NVBUFFER_COMPOSITE` = 1 Flag to set for composition.	`NVBUFSURF_TRANSFORM_COMPOSITE` Specifies a flag to describe the requested compositing operation.
`NVBUFFER_BLEND` = 1 << 1 Flag to set for blending.	
`NVBUFFER_COMPOSITE_FILTER` = 1 << 2 Composition flag to set filter type.	`NVBUFSURF_TRANSFORM_COMPOSITE_FILTER` Specifies a composite to set the filter type.

Buffer Color Formats	
NvBufferColorFormat (`nvbuf_utils.h`)	**NvBufSurfaceColorFormat** (`nvbufsurftransform.h`)
`NvBufferColorFormat_YUV420` BT.601 color space - YUV420 multi-planar.	`NVBUF_COLOR_FORMAT_YUV420` Specifies BT.601 color space - YUV420 multi-planar.
`NvBufferColorFormat_YVU420` BT.601 color space - YUV420 multi-planar.	`NVBUF_COLOR_FORMAT_YVU420` Specifies BT.601 color space - YVU420 multi-planar.
`NvBufferColorFormat_YUV422` BT.601 color space - YUV422 multi-planar.	`NVBUF_COLOR_FORMAT_YUV422` Specifies BT.601 color space - YUV422 multi-planar.
`NvBufferColorFormat_YUV420_ER` BT.601 color space - YUV420 ER multi-planar.	`NVBUF_COLOR_FORMAT_YUV420_ER` Specifies BT.601 color space - YUV420 ER multi-planar.
`NvBufferColorFormat_YVU420_ER` BT.601 color space - YVU420 ER multi-planar.	`NVBUF_COLOR_FORMAT_YVU420_ER` Specifies BT.601 color space - YVU420 ER multi-planar.
`NvBufferColorFormat_NV12` BT.601 color space - Y/CbCr 4:2:0 multi-planar.	`NVBUF_COLOR_FORMAT_NV12` Specifies BT.601 color space - Y/CbCr 4:2:0 multi-planar.
`NvBufferColorFormat_NV12_ER` BT.601 color space - Y/CbCr ER 4:2:0 multi-planar.	`NVBUF_COLOR_FORMAT_NV12_ER` Specifies BT.601 color space - Y/CbCr ER 4:2:0 multi-planar.
`NvBufferColorFormat_NV21` BT.601 color space - Y/CbCr 4:2:0 multi-planar.	`NVBUF_COLOR_FORMAT_NV21` Specifies BT.601 color space - Y/CbCr 4:2:0 multi-planar.
`NvBufferColorFormat_NV21_ER` BT.601 color space - Y/CbCr ER 4:2:0 multi-planar.	`NVBUF_COLOR_FORMAT_NV21_ER` Specifies BT.601 color space - Y/CbCr ER 4:2:0 multi-planar.

NvBufferColorFormat (nvbuf_utils.h)	**NvBufSurfaceColorFormat** (nvbufsurftransform.h)
NvBufferColorFormat_UYVY BT.601 color space - YUV 4:2:2 planar.	NVBUF_COLOR_FORMAT_UYVY Specifies BT.601 color space - YUV 4:2:2 planar.
NvBufferColorFormat_UYVY_ER BT.601 color space - YUV ER 4:2:2 planar	NVBUF_COLOR_FORMAT_UYVY_ER Specifies BT.601 color space - YUV ER 4:2:2 planar.
NvBufferColorFormat_VYUY BT.601 color space - YUV 4:2:2 planar.	NVBUF_COLOR_FORMAT_VYUY Specifies BT.601 color space - YUV 4:2:2 planar.
NvBufferColorFormat_VYUY_ER BT.601 color space - YUV ER 4:2:2 planar.	NVBUF_COLOR_FORMAT_VYUY_ER Specifies BT.601 color space - YUV ER 4:2:2 planar.
NvBufferColorFormat_YUYV BT.601 color space - YUV 4:2:2 planar.	NVBUF_COLOR_FORMAT_YUYV Specifies BT.601 color space - YUV 4:2:2 planar.
NvBufferColorFormat_YUYV_ER BT.601 color space - YUV ER 4:2:2 planar.	NVBUF_COLOR_FORMAT_YUYV_ER Specifies BT.601 color space - YUV ER 4:2:2 planar.
NvBufferColorFormat_YVYU BT.601 color space - YUV 4:2:2 planar.	NVBUF_COLOR_FORMAT_YVYU Specifies BT.601 color space - YUV 4:2:2 planar.
NvBufferColorFormat_YVYU_ER BT.601 color space - YUV ER 4:2:2 planar.	NVBUF_COLOR_FORMAT_YVYU_ER Specifies BT.601 color space - YUV ER 4:2:2 planar.
NvBufferColorFormat_ABGR32 Legacy RGBA color space - BGRA-8-8-8-8 planar.	NVBUF_COLOR_FORMAT_RGBA Specifies RGBA-8-8-8-8 single plane.
NvBufferColorFormat_XRGB32 Legacy RGBA color space - XRGB-8-8-8-8 planar.	NVBUF_COLOR_FORMAT_BGRx Specifies BGRx-8-8-8-8 single plane.
NvBufferColorFormat_ARGB32 Legacy RGBA color space - ARGB-8-8-8-8 planar.	NVBUF_COLOR_FORMAT_BGRA Specifies BGRA-8-8-8-8 single plane.
NvBufferColorFormat_NV12_10LE BT.601 color space - Y/CbCr 4:2:0 10-bit multi-planar.	NVBUF_COLOR_FORMAT_NV12_10LE Specifies BT.601 color space - Y/CbCr 4:2:0 10-bit multi-planar.
NvBufferColorFormat_NV12_10LE_709 BT.709 color space - Y/CbCr 4:2:0 10-bit multi-planar.	NVBUF_COLOR_FORMAT_NV12_10LE_709 Specifies BT.709 color space - Y/CbCr 4:2:0 10-bit multi-planar.
NvBufferColorFormat_NV12_10LE_709_ER BT.709_ER color space - Y/CbCr 4:2:0 10-bit multi-planar.	NVBUF_COLOR_FORMAT_NV12_10LE_709_ER Specifies BT.709 color space - Y/CbCr ER 4:2:0 10-bit multi-planar.
NvBufferColorFormat_NV12_10LE_2020 BT.2020 color space - Y/CbCr 4:2:0 10-bit multi-planar.	NVBUF_COLOR_FORMAT_NV12_10LE_2020 Specifies BT.2020 color space - Y/CbCr 4:2:0 10-bit multi-planar.

NvBufferColorFormat (nvbuf_utils.h)	**NvBufSurfaceColorFormat** (nvbufsurftransform.h)
NvBufferColorFormat_NV21_10LE BT.601 color space - Y/CrCb 4:2:0 10-bit multi-planar.	NVBUF_COLOR_FORMAT_NV21_10LE Specifies BT.601 color space - Y/CrCb 4:2:0 10-bit multi-planar.
NvBufferColorFormat_NV12_12LE BT.601 color space - Y/CbCr 4:2:0 12-bit multi-planar.	NVBUF_COLOR_FORMAT_NV12_12LE Specifies BT.601 color space - Y/CbCr 4:2:0 12-bit multi-planar.
NvBufferColorFormat_NV12_12LE_2020 BT.2020 color space - Y/CbCr 4:2:0 12-bit multi-planar.	NVBUF_COLOR_FORMAT_NV12_12LE_2020 Specifies BT.2020 color space - Y/CbCr 4:2:0 12-bit multi-planar.
NvBufferColorFormat_NV21_12LE BT.601 color space - Y/CrCb 4:2:0 12-bit multi-planar.	NVBUF_COLOR_FORMAT_NV21_12LE Specifies BT.601 color space - Y/CrCb 4:2:0 12-bit multi-planar.
NvBufferColorFormat_YUV420_709 BT.709 color space - YUV420 multi-planar.	NVBUF_COLOR_FORMAT_YUV420_709 Specifies BT.709 color space - YUV420 multi-planar.
NvBufferColorFormat_YUV420_709_ER BT.709 color space - YUV420 ER multi-planar.	NVBUF_COLOR_FORMAT_YUV420_709_ER Specifies BT.709 color space - YUV420 ER multi-planar.
NvBufferColorFormat_NV12_709 BT.709 color space - Y/CbCr 4:2:0 multi-planar.	NVBUF_COLOR_FORMAT_NV12_709 Specifies BT.709 color space - Y/CbCr 4:2:0 multi-planar.
NvBufferColorFormat_NV12_709_ER BT.709 color space - Y/CbCr ER 4:2:0 multi-planar.	NVBUF_COLOR_FORMAT_NV12_709_ER Specifies BT.709 color space - Y/CbCr ER 4:2:0 multi-planar.
NvBufferColorFormat_YUV420_2020 BT.2020 color space - YUV420 multi-planar.	NVBUF_COLOR_FORMAT_YUV420_2020 Specifies BT.2020 color space - YUV420 multi-planar.
NvBufferColorFormat_NV12_2020 BT.2020 color space - Y/CbCr 4:2:0 multi-planar.	NVBUF_COLOR_FORMAT_NV12_2020 Specifies BT.2020 color space - Y/CbCr 4:2:0 multi-planar.
NvBufferColorFormat_YUV444 BT.601 color space - YUV444 multi-planar.	NVBUF_COLOR_FORMAT_YUV444 Specifies BT.601 color space - YUV444 multi-planar.
NvBufferColorFormat_SignedR16G16 Optical flow/	NVBUF_COLOR_FORMAT_SIGNED_R16G16 Specifies color format for packed 2 signed shorts.
NvBufferColorFormat_A32 Optical flow SAD calculation Buffer format.	NVBUF_COLOR_FORMAT_A32 Specifies Optical flow SAD calculation Buffer format/
NvBufferColorFormat_GRAY8 8-bit grayscale.	NVBUF_COLOR_FORMAT_GRAY8 Specifies 8-bit GRAY scale - single plane.

NvBufferColorFormat (`nvbuf_utils.h`)	**NvBufSurfaceColorFormat** (`nvbufsurftransform.h`)
`NvBufferColorFormat_NV16` BT.601 color space - Y/CbCr 4:2:2 multi-planar.	`NVBUF_COLOR_FORMAT_NV16` Specifies BT.601 color space - Y/CbCr 4:2:2 multi-planar.
`NvBufferColorFormat_NV16_10LE` BT.601 color space - Y/CbCr 4:2:2 10-bit semi-planar.	`NVBUF_COLOR_FORMAT_NV16_10LE` Specifies BT.601 color space - Y/CbCr 4:2:2 10-bit semi-planar.
`NvBufferColorFormat_NV24` BT.601 color space - Y/CbCr 4:4:4 multi-planar.	`NVBUF_COLOR_FORMAT_NV24` Specifies BT.601 color space - Y/CbCr 4:4:4 multi-planar.
`NvBufferColorFormat_NV24_10LE` BT.601 color space - Y/CrCb 4:4:4 10-bit multi-planar.	`NVBUF_COLOR_FORMAT_NV24_10LE` Specifies BT.601 color space - Y/CrCb 4:4:4 10-bit multi-planar.
`NvBufferColorFormat_NV16_ER` BT.601_ER color space - Y/CbCr 4:2:2 multi-planar.	`NVBUF_COLOR_FORMAT_NV16_ER` Specifies BT.601_ER color space - Y/CbCr 4:2:2 multi-planar.
`NvBufferColorFormat_NV24_ER` BT.601_ER color space - Y/CbCr 4:4:4 multi-planar.	`NVBUF_COLOR_FORMAT_NV24_ER` Specifies BT.601_ER color space - Y/CbCr 4:4:4 multi-planar.
`NvBufferColorFormat_NV16_709` BT.709 color space - Y/CbCr 4:2:2 multi-planar.	`NVBUF_COLOR_FORMAT_NV16_709` Specifies BT.709 color space - Y/CbCr 4:2:2 multi-planar.
`NvBufferColorFormat_NV24_709` BT.709 color space - Y/CbCr 4:4:4 multi-planar.	`NVBUF_COLOR_FORMAT_NV24_709` Specifies BT.709 color space - Y/CbCr 4:4:4 multi-planar.
`NvBufferColorFormat_NV16_709_ER` BT.709_ER color space - Y/CbCr 4:2:2 multi-planar.	`NVBUF_COLOR_FORMAT_NV16_709_ER` Specifies BT.709_ER color space - Y/CbCr 4:2:2 multi-planar.
`NvBufferColorFormat_NV24_709_ER` BT.709_ER color space - Y/CbCr 4:4:4 multi-planar.	`NVBUF_COLOR_FORMAT_NV24_709_ER` Specifies BT.709_ER color space - Y/CbCr 4:4:4 multi-planar.
`NvBufferColorFormat_NV24_10LE_709` BT.709 color space - Y/CbCr 10 bit 4:4:4 multi-planar.	`NVBUF_COLOR_FORMAT_NV24_10LE_709` Specifies BT.709 color space - Y/CbCr 10 bit 4:4:4 multi-planar.
`NvBufferColorFormat_NV24_10LE_709_ER` BT.709 ER color space - Y/CbCr 10 bit 4:4:4 multi-planar.	`NVBUF_COLOR_FORMAT_NV24_10LE_709_ER` Specifies BT.709 ER color space - Y/CbCr 10 bit 4:4:4 multi-planar.
`NvBufferColorFormat_NV24_10LE_2020` BT.2020 color space – Y/CbCr 10 bit 4:4:4 multi-planar.	`NVBUF_COLOR_FORMAT_NV24_10LE_2020` Specifies BT.2020 color space – Y/CbCr 10 bit 4:4:4 multi-planar.

NvBufferColorFormat (`nvbuf_utils.h`)	**NvBufSurfaceColorFormat** (`nvbufsurftransform.h`)
`NvBufferColorFormat_NV24_12LE_2020` BT.2020 color space - Y/CbCr 12 bit 4:4:4 multi-planar.	`NVBUF_COLOR_FORMAT_NV24_12LE_2020` Specifies BT.2020 color space - Y/CbCr 12 bit 4:4:4 multi-planar.
`NvBufferColorFormat_RGBA_10_10_10_2_709` Non-linear RGB BT.709 color space - RGBA-10-10-10-2 planar.	`NVBUF_COLOR_FORMAT_RGBA_10_10_10_2_709` Specifies Non-linear RGB BT.709 color space - RGBA-10-10-10-2 planar.
`NvBufferColorFormat_RGBA_10_10_10_2_2020` Non-linear RGB BT.2020 color space - RGBA-10-10-10-2 planar.	`NVBUF_COLOR_FORMAT_RGBA_10_10_10_2_2020` Specifies Non-linear RGB BT.2020 color space - RGBA-10-10-10-2 planar.
`NvBufferColorFormat_BGRA_10_10_10_2_709` Non-linear RGB BT.709 color space - BGRA-10-10-10-2 planar.	`NVBUF_COLOR_FORMAT_BGRA_10_10_10_2_709` Specifies Non-linear RGB BT.709 color space - BGRA-10-10-10-2 planar.
`NvBufferColorFormat_BGRA_10_10_10_2_2020` Non-linear RGB BT.2020 color space - BGRA-10-10-10-2 planar.	`NVBUF_COLOR_FORMAT_BGRA_10_10_10_2_2020` Specifies Non-linear RGB BT.2020 color space - BGRA-10-10-10-2 planar.
`NvBufferColorFormat_Invalid` Invalid color format.	`NVBUF_COLOR_FORMAT_INVALID` Specifies an invalid color format.
	`NVBUF_COLOR_FORMAT_ARGB` Specifies ARGB-8-8-8-8 single plane.
	`NVBUF_COLOR_FORMAT_ABGR` Specifies ABGR-8-8-8-8 single plane.
	`NVBUF_COLOR_FORMAT_RGBx` Specifies RGBx-8-8-8-8 single plane.
	`NVBUF_COLOR_FORMAT_xRGB` Specifies xRGB-8-8-8-8 single plane.
	`NVBUF_COLOR_FORMAT_xBGR` Specifies xBGR-8-8-8-8 single plane.
	`NVBUF_COLOR_FORMAT_RGB` Specifies RGB-8-8-8 single plane.
	`NVBUF_COLOR_FORMAT_BGR` Specifies BGR-8-8-8 single plane.
	`NVBUF_COLOR_FORMAT_NV12_10LE_ER` Specifies BT.601 color space - Y/CbCr ER 4:2:0 10-bit multi-planar.
	`NVBUF_COLOR_FORMAT_R8_G8_B8` Specifies RGB- unsigned 8-bit multiplanar plane.

NvBufferColorFormat (`nvbuf_utils.h`)	**NvBufSurfaceColorFormat** (`nvbufsurftransform.h`)
	`NVBUF_COLOR_FORMAT_B8_G8_R8` Specifies BGR- unsigned 8-bit multiplanar plane.
	`NVBUF_COLOR_FORMAT_R32F_G32F_B32F` Specifies RGB-32bit Floating point multiplanar plane.
	`NVBUF_COLOR_FORMAT_B32F_G32F_R32F` Specifies BGR-32bit Floating point multiplanar plane.

Ported Data Type Definitions

This section lists the data types ported from `nvbuf_utils.h`.

nvbufsurface.h Data Types

Buffer Handle

NvUtils replaces the `dmabuf_fd` buffer handle with the `NvBufSurface` structure. This structure is used as a buffer container or handle.

- `dmabuf_fd` holds a hardware DMA (Direct Memory Access) buffer handle.
- `NvBufSurface` holds information about batch buffers.

dmabuf_fd (`nvbuf_utils.h`)	**NvBufSurface** (`nvbufsurftransform.h`)
`int dmabuf_fd` Holds a hardware DMA (Direct Memory Access) buffer handle.	`uint32_t gpuId` Holds a GPU ID. Valid only for a multi-GPU system.
	`uint32_t batchSize` Holds the batch size.
	`uint32_t numFilled` Holds the number valid and filled buffers. Initialized to zero when an instance of the structure is created.
	`bool isContiguous` Holds an "is contiguous" flag. If set, memory allocated for the batch is contiguous.
	`NvBufSurfaceMemType memType` Holds type of memory for buffers in the batch.
	`NvBufSurfaceParams *surfaceList` Holds a pointer to an array of batched buffers.
	`void * _reserved[STRUCTURE_PADDING]` Holds structure padding

Chroma Subsampling

NvUtils replaces the `NvBufferChromaSubsamplingParams` structure with `NvBufSurfaceChromaSubsamplingParams`. Both hold the chroma subsampling parameters.

NvBufferChromaSubsamplingParams (`nvbuf_utils.h`)	**NvBufSurfaceChromaSubsamplingParams** (`nvbufsurftransform.h`)
`uint8_t chromaLocHoriz` Location settings.	`uint8_t chromaLocHoriz` Location settings.
`uint8_t chromaLocVert` Location settings.	`uint8_t chromaLocVert` Location settings.

Hardware Buffer Creation

NvUtils replaces the `NvBufferCreateParams` structure with `NvBufSurfaceAllocateParams` and `NvBufSurfaceCreateParams`, which hold the input parameters for hardware buffer creation.

NvBufferCreateParams (`nvbuf_utils.h`)	**NvBufSurfaceAllocateParams** (`nvbufsurftransform.h`)
`uint8_t chromaLocHoriz` Location settings.	`uint8_t chromaLocHoriz` Location settings.
	`NvBufSurfaceCreateParams params` Hold legacy NvBufSurface creation parameters.
	`NvBufSurfaceDisplayScanFormat displayscanformat` Display scan format.
	`NvBufSurfaceChromaSubsamplingParams chromaSubsampling` Chroma Subsampling parameters.
`NvBufferTag nvbuf_tag` Tag to associate with the buffer.	`NvBufSurfaceTag memtag` Components tag to be used for memory allocation.
	`void * _reserved[STRUCTURE_PADDING];` Reserved for padding.

NvBufferCreateParams (`nvbuf_utils.h`)	**NvBufSurfaceCreateParams** (`nvbufsurftransform.h`)
	`uint32_t gpuId` Holds the GPU ID. Valid only for a multi-GPU system.

NvBufferCreateParams (nvbuf_utils.h)	NvBufSurfaceCreateParams (nvbufsurftransform.h)
`int32_t width` Width of the buffer.	**`uint32_t width`** Holds the width of the buffer.
`int32_t height` Height of the buffer.	**`uint32_t height`** Holds the height of the buffer.
`int32_t memsize` Size of the memory. Applicable for `NvBufferPayload_MemHandle`.	`uint32_t size` Holds the amount of memory to be allocated. Optional; if set, all other parameters (width, height, etc.) are ignored.
	`bool isContiguous` Holds a "contiguous memory" flag. If set, contiguous memory is allocated for the batch. Valid only for CUDA memory types.
`NvBufferColorFormat colorFormat` Color format of the buffer.	**`NvBufSurfaceColorFormat colorFormat`** Holds the color format of the buffer.
`NvBufferLayout layout` layout of the buffer.	**`NvBufSurfaceLayout layout`** Holds the surface layout. May be Block Linear (BL) or Pitch Linear (PL). For a dGPU, only PL is valid.
`NvBufferPayloadType payloadType` Payload type of the buffer.	**`NvBufSurfaceMemType memType`** Holds the type of memory to be allocated.

Hardware Buffer Parameters

NvUtils replaces the `NvBufferParams` structure with `NvBufSurfaceParams`, and `NvBufferParamsEx` with `NvBufSurfaceParamsEx`. `NvBufSurfaceParams` holds the buffer parameters, and `NvBufSurfaceParamsEx` holds the extended parameters for a hardware buffer.

Note: Some of the fields of `NvBufferParams` can be retrieved directly from `NvBufSurface`.

NvUtils does not require explicit calls to `NvBufferGetParams` and `NvBufferGetParamsEx` to get updated buffer parameters because you can retrieve all of the parameters using the `NvBufSurface` handle.

NvBufferParams (nvbuf_utils.h)	NvBufSurfaceParams (nvbufsurface.h)
	`uint32_t width` Holds the width of the buffer.
	`uint32_t height` Holds the height of the buffer.

NvBufferParams (`nvbuf_utils.h`)	**NvBufSurfaceParams** (`nvbufsurface.h`)
	`uint32_t pitch` Holds the pitch of the buffer.
`NvBufferColorFormat pixel_format` Video format type of hardware buffer.	`NvBufSurfaceColorFormat colorFormat` Holds the color format of the buffer.
`uint32_t layout[MAX_NUM_PLANES]` Layout type of each plane of the hardware buffer.	`NvBufSurfaceLayout layout` Holds the layout, BL or PL. For dGPU, only PL is valid.
`uint32_t dmabuf_fd` Holds the DMABUF file descriptor of the hardware buffer.	`uint64_t bufferDesc` Holds a DMABUF file descriptor. Valid only for `NVBUF_MEM_SURFACE_ARRAY` and `NVBUF_MEM_HANDLE` type memory.
`int32_t memsize` Size of the memory. Applicable to `NvBufferPayload_MemHandle`.	`uint32_t dataSize` Holds the amount of allocated memory.
`void *nv_buffer` Pointer to hardware buffer memory.	`void * dataPtr` Holds a pointer to allocated memory.
	`NvBufSurfacePlaneParams planeParams` Holds planewise information (width, height, pitch, offset, etc.).
	`NvBufSurfaceMappedAddr mappedAddr` Holds pointers to mapped buffers. Initialized to NULL when the structure is created.
	`NvBufSurfaceParamsEx *paramex` Pointers to extended parameters of a single buffer in the batch.
	`void * _reserved[STRUCTURE_PADDING - 1]` Reserved for padding.
`NvBufferPayloadType payloadType` payload type of the buffer.	See the `NvBufSurface` structure, member `memType`.
`uint32_t nv_buffer_size` Size of the hardware buffer.	

NvBufferParams (`nvbuf_utils.h`)	**NvBufSurfacePlaneParams** (`nvbufsurface.h`)
`uint32_t num_planes` Number of planes in the hardware buffer.	`uint32_t num_planes` Holds the number of planes.

NvBufferParams (`nvbuf_utils.h`)	**NvBufSurfacePlaneParams** (`nvbufsurface.h`)
`uint32_t width[MAX_NUM_PLANES]` Width of each plane in the hardware buffer.	`uint32_t width[NVBUF_MAX_PLANES]` Holds the widths of planes.
`uint32_t height[MAX_NUM_PLANES]` Height of each plane in the hardware buffer.	`uint32_t height[NVBUF_MAX_PLANES]` Holds the heights of planes.
`uint32_t pitch[MAX_NUM_PLANES]` Pitch of each plane in the hardware buffer.	`uint32_t pitch[NVBUF_MAX_PLANES]` Holds the pitches of planes in bytes.
`uint32_t offset[MAX_NUM_PLANES]` Memory offset values of each video plane in the hardware buffer.	`uint32_t offset[NVBUF_MAX_PLANES]` Holds the offsets of planes in bytes.
`uint32_t psize[MAX_NUM_PLANES]` Size of each video plane in the hardware buffer.	`uint32_t psize[NVBUF_MAX_PLANES]` Holds the sizes of planes in bytes.
	`uint32_t bytesPerPix[NVBUF_MAX_PLANES]` Holds the number of bytes occupied by a pixel in each plane.
	`void * _reserved[STRUCTURE_PADDING * NVBUF_MAX_PLANES]` Reserved for padding

NvBufferParamsEx (`nvbuf_utils.h`)	**NvBufSurfaceParamsEx** (`nvbufsurface.h`)
`int32_t startofvaliddata` Offset in bytes from the start of the buffer to the first valid byte. Applicale to `NvBufferPayload_MemHandle`,	`int32_t startofvaliddata` Offset in bytes from the start of the buffer to the first valid byte. Applicable to `NVBUF_MEM_HANDLE`.
`int32_t sizeofvaliddatainbytes` Size of valid data from first to last valid byte. Applicable to `NvBufferPayload_MemHandle`.	Size of valid data in bytes. Applicable to `NVBUF_MEM_HANDLE`. `int32_t sizeofvaliddatainbytes`
`NvBufferChromaSubsamplingParams chromaSubsampling` Chroma subsampling parameters.	`NvBufSurfaceChromaSubsamplingParams chromaSubsampling` Chroma subsampling parameters. Applicable to `NVBUF_MEM_SURFACE_ARRAY`.
`bool is_protected` Get buffer VPR information.	`bool is_protected` Get buffer VPR information.

NvBufferParamsEx (`nvbuf_utils.h`)	**NvBufSurfaceParamsEx** (`nvbufsurface.h`)
	`NvBufSurfacePlaneParamsEx planeParamsex` Plane-wise extended information.
`void *reserved` Reserved field.	`void * _reserved[STRUCTURE_PADDING]` Reserved for padding.

NvBufferParamsEx (`nvbuf_utils.h`)	**NvBufSurfacePlaneParamsEx** (`nvbufsurface.h`)
`NvBufferDisplayScanFormat scanformat[MAX_NUM_PLANES]` Display scan format, progressive or interlaced.	`NvBufSurfaceDisplayScanFormat scanformat[NVBUF_MAX_PLANES]` Display scan format, progressive or interlaced.
`uint32_t secondfieldoffset[MAX_NUM_PLANES]` Offset of the second field in an interlaced buffer.	`uint32_t secondfieldoffset[NVBUF_MAX_PLANES]` Offset of the second field in an interlaced buffer.
`uint32_t blockheightlog2[MAX_NUM_PLANES]` Block height of the planes in a blockLinear layout hardware buffer.	`uint32_t blockheightlog2[NVBUF_MAX_PLANES]` Block height of the planes in a blockLinear layout buffer.
`uint32_t physicaladdress[MAX_NUM_PLANES]` Physical address of allocated planes.	`uint32_t physicaladdress[NVBUF_MAX_PLANES]` Physical address of allocated planes.
`uint64_t flags[MAX_NUM_PLANES]` Flags associated with planes.	`uint64_t flags[NVBUF_MAX_PLANES]` Flags associated with planes.
	`void * _reserved[STRUCTURE_PADDING * NVBUF_MAX_PLANES]` Reserved for padding.
`NvBufferParams params` NvBuffer basic parameters.	See the `NvBufSurface` structure.
`void *payloadmetaInfo` Metadata associated with the hardware buffer.	
`NvBufferSyncObj syncobj` Buffer sync point object parameters.	

nvbufsurftransform.h Data Types

Buffer Sync Point Object

NvUtils replaces the `NvBufferSyncObj` structure with `NvBufSurfTransformSyncObj`. Both structures hold information about synchronization objects for asynchronous transform or composite APIs.

NvBufferSyncObj (`nvbuf_utils.h`)	**NvBufSurfTransformSyncObj** (`nvbufsurftransform.h`)
`NvBufferSyncObjParams insyncobj[NVBUF_MAX_SYNCOBJ_PARAMS]` Array of input sync objects.	`union { cudaEvent_t cuEvent, NvRmFence vicEvent }` Either a CUDA event or a `NvRmFence`.
`uint32_t num_insyncobj` Number of input sync object instances.	`uint32_t eventType` Type of synch barrier: `cudaEvent` or `NvRmFence`,
`NvBufferSyncObjParams outsyncobj` Output sync object.	
`uint32_t use_outsyncobj` Flag to use `outsyncobj`.	
	`bool waitDone` Flag to check the status.

NvBufferSyncObjParams (`nvbuf_utils.h`)	**NvBufSurfTransformSyncObj** (`nvbufsurftransform.h`)

`uint32_t syncpointID;`
`uint32_t value;`
Holds parameters for a buffer sync point object, consisting of a [sync point ID, value] pair. A client can use this object to describe an event that it wants to wait for.	

Buffer Composition Background

NvUtils replaces the structure `NvBufferCompositeBackground` with `NvBufSurfTransform_ColorParams`. Both stuctures hold the composition background RGB

color value.

NvBufferCompositeBackground (`nvbuf_utils.h`)	**NvBufSurfTransform_ColorParams** (`nvbufsurftransform.h`)
`float r` Background color value for r.	`double red` Holds the red component of color. Value must be in the range 0.0-1.0.
`float g` Background color value for g.	`double green` Holds the green component of color. Value must be in the range 0.0-1.0.
`float b` Background color value for b.	`double blue` Holds the blue component of color. Value must be in the range 0.0-1.0.
	`double alpha` Holds the alpha component of color. Value must be in the range 0.0-1.0.

Rectangular Coordinates

NvUtils replaces the structure `NvBufferRect` with `NvBufSurfTransformRect`.

NvBufferRect (`nvbuf_utils.h`)	**NvBufSurfTransformRect** (`nvbufsurftransform.h`)
`uint32_t top` Rectangle top.	`uint32_t top` Holds the rectangle top.
`uint32_t left` Rectangle left.	`uint32_t left` Holds the rectangle left side
`uint32_t width` Rectangle width.	`uint32_t width` Holds the rectangle width.
`uint32_t height` Rectangle height.	`uint32_t height` Holds the rectangle height.

Composite Parameters

NvUtils replaces the `NvBufferCompositeParams` structure with `NvBufSurfTransformCompositeParams` and `NvBufSurfTransformCompositeBlendParams`.

`NvBufSurfTransformCompositeParams` holds composite parameters for a composite call.

`NvBufSurfTransformCompositeBlendParams` holds composite blend parameters for a composite blender call.

Note: For composition, `NvBufSurfTransformMultiInputBufComposite()` must be used with `NvBufSurfTransformCompositeParams()`. Blending functions are currently not available in the API.

NvBufferCompositeParams (`nvbuf_utils.h`)	**NvBufSurfTransformCompositeParams** (`nvbufsurftransform.h`)
`uint32_t composite_flag` Indicates which of the composition and blending parameters are valid.	`uint32_t composite_flag` Holds a flag that indicates which composition parameters are valid.
`uint32_t input_buf_count` Number of input buffers to be composited.	`uint32_t input_buf_count` Holds the number of input buffers to be composited.
`NvBufferRect src_comp_rect[MAX_COMPOSITE_FRAME]` Source rectangle coordinates of input buffers for composition.	`NvBufSurfTransformRect *src_comp_rect` Holds source rectangle coordinates of input buffers for compositing.
`NvBufferRect dst_comp_rect[MAX_COMPOSITE_FRAME]` Destination rectangle coordinates of input buffers for composition.	`NvBufSurfTransformRect *dst_comp_rect` Holds destination rectangle coordinates of input buffers for compositing.
`NvBufferTransform_Filter composite_filter[MAX_COMPOSITE_FRAME]` Filters to use for composition.	`NvBufSurfTransform_Inter composite_filter` Holds a composite filter.

NvBufferCompositeParams (`nvbuf_utils.h`)	**NvBufSurfTransformCompositeBlendParams** (`nvbufsurftransform.h`)
	`uint32_t composite_blend_flag` Holds a flag that indicates which composition parameters are valid.
	`uint32_t input_buf_count` Holds the number of input buffers to be composited.
	`NvBufSurfTransform_Inter composite_blend_filter` Holds a blend/composite filter applicable only

NvBufferCompositeParams (`nvbuf_utils.h`)	**NvBufSurfTransformCompositeBlendParams** (`nvbufsurftransform.h`)
`NvBufferCompositeBackground composite_bgcolor` Background color values for composition.	`NvBufSurfTransform_ColorParams *color_bg` Holds a background color list for blending. If the background buffer is absent if NULL, this list is not used because the background buffer is expected to be NULL. If blending is required it must be done with a static color.
`float dst_comp_rect_alpha[MAX_COMPOSITE_FRAME]` Alpha values of input buffers for the blending.	
	`uint32_t *perform_blending` Holds a list of Boolean flags indicating whether particular buffers are to be blended. If NULL, all buffers are blended. If not NULL, the list must contain at least `numFilled` elements, each with value 0 or 1.
`NvBufferSession session` Session to be used for composition. If NULL, the default session is used.	

Buffer Transform Parameters

NvUtils replaces the `NvBufferTransformParams` structure with `NvBufSurfTransformParams`.

Both structures hold parameters for buffer transformation functions.

Another structure, `NvBufSurfTransformConfigParams`, holds user-defined session parameters for transform and composite.

NvBufferTransformParams (`nvbuf_utils.h`)	**NvBufSurfTransformParams** (`nvbufsurftransform.h`)
`uint32_t transform_flag` Flag to indicate which of the transform parameters are valid.	`uint32_t transform_flag` Holds a flag that indicates which transform parameters are valid.
`NvBufferTransform_Flip transform_flip` Flip method.	`NvBufSurfTransform_Flip transform_flip` Holds the flip method.
`NvBufferTransform_Filter transform_filter` Transform filter.	`NvBufSurfTransform_Inter transform_filter` Holds a transform filter.

NvBufferTransformParams (`nvbuf_utils.h`)	**NvBufSurfTransformParams** (`nvbufsurftransform.h`)
`NvBufferRect src_rect` Source rectangle coordinates for crop operation.	`NvBufSurfTransformRect *src_rect` Holds a pointer to a list of source rectangle coordinates for a crop operation.
`NvBufferRect dst_rect` Destination rectangle coordinates for crop operation.	`NvBufSurfTransformRect *dst_rect` Holds a pointer to list of destination rectangle coordinates for a crop operation.
`NvBufferSession session` `NvBufferSession` to be used for transform. If NULL, the default session is used.	

NvBufferTransformParams (`nvbuf_utils.h`)	**NvBufSurfTransformConfigParams** (`nvbufsurftransform.h`)
	`NvBufSurfTransform_Compute compute_mode` Holds the mode of operation: VIC (Jetson) or GPU (iGPU + dGPU). If VIC is configured, \a gpu_id is ignored.
	`int32_t gpu_id` Holds the GPU ID to be used for processing.
	`cudaStream_t cuda_stream` User configured stream to be used. If NULL, the default stream is used. Ignored if VIC is used.

Interface APIs Comparison for Migration

Buffer Allocation related Interface APIs

nvbuf_utils:

NvBufferCreateEx

```
int
NvBufferCreateEx (int *dmabuf_fd, NvBufferCreateParams *input_params
```

Allocates a hardware buffer

Parameters:

* dmabuf_fd [out] - Returns the DMABUF FD of the hardware buffer.

* input_params [in] - Input parameters for hardware buffer creation.

Returns – 0 for success, -1 for failure

NvBufferCreateInterlace

```
int
NvBufferCreateInterlace (int *dmabuf_fd, NvBufferCreateParams *input_params)
```

Allocates a hardware buffer for interlace scan format

Parameters:

* dmabuf_fd [out] - Returns the DMABUF FD of the hardware buffer.

* input_params [in] - Input parameters for hardware buffer creation.

Returns – 0 for success, -1 for failure

NvBufferCreateWithChromaLoc

```
int
NvBufferCreateWithChromaLoc (int *dmabuf_fd, NvBufferCreateParams *input_params,
                                        NvBufferChromaSubsamplingParams *chromaSubsampling)
```

Allocates a hardware buffer with a given chroma subsampling location.

Parameters:

* dmabuf_fd [out] - Returns the DMABUF FD of the hardware buffer.

* input_params [in] - Input parameters for hardware buffer creation.

* chromaSubsampling [in] - Chroma location parameters.

Returns – 0 for success, -1 for failure

➪ **Above nvbuf_utils buffer allocation related APIs can be migrated using below nvutils API.**

nvutils:

NvBufSurfaceAllocate

```
int
NvBufSurfaceAllocate (NvBufSurface **surf, uint32_t batchSize,
                                NvBufSurfaceAllocateParams*paramsext)
```

Allocate batch of buffers. (Using extended buffer allocation parameters)

Allocates memory for batchSize buffers and returns in *surf a pointer to allocated NvBufSurface.

params structure should have allocation parameters of single buffer. If size field in params is set, buffer of that size will be allocated and all other parameters (w, h, color format etc.) will be ignored.

Use NvBufSurfaceDestroy() to free all the resources.

NOTE: Refer "**NvBufSurfaceAllocateParams**" structure in section Hardware Buffer Creation for comparison with legacy nvbuf_utils used hardware buffer creation related input parameters.

“bufferDesc” field of “NvBufSurfaceParams” of NvBufSurface holds a *DMABUF FD* which is same as dmabuf_fd allocated using above all nvbuf_utils buffer allocation interface APIs .

Parameters:

* surf [out] - Pointer to allocated batched buffers.

* batchSize [in] - Batch size of buffers.

* paramsext [in] - Pointer to NvBufSurfaceAllocateParams structure.

Returns 0 for success, -1 for failure.

Buffer Destroy related Interface APIs

nvbuf_utils:

```
int
NvBufferDestroy (int dmabuf_fd)
```

Destroys a hardware buffer.

Parameters:

* dmabuf_fd [in] - Specifies the `dmabuf_fd` `hw_buffer` to destroy

Returns 0 for success, -1 for failure.

⇨ Above nvbuf_utils buffer deallocation related APIs can be migrated using below nvutils API.

nvutils:

```
int
NvBufSurfaceDestroy (NvBufSurface *surf)
```

Free the batched buffers previously allocated through NvBufSurfaceCreate.

Parameters:

* surf [in] - A pointer to an NvBufSurface to be freed.

Returns 0 for success, -1 for failure.

API to sync the hardware memory cache for the CPU

nvbuf_utils:

```
int
NvBufferMemSyncForCpu (int dmabuf_fd, unsigned int plane, void **pVirtAddr)
```

Syncs the hardware memory cache for the CPU. Refer NvBufferMemMap for the purpose of the function

* dmabuf_fd [in] - DMABUF FD of buffer.

* plane[in] - Video frame plane.

* pVirtAddr [in] - Virtual Address pointer of the memory-mapped plane.

Returns 0 for success, -1 for failure.

- ⇨ **Above nvbuf_utils API to sync the hardware memory cache for cpu access can be migrated using below nvutils API.**

nvutils:

```
int
NvBufSurfaceSyncForCpu (NvBufSurface *surf, int index, int plane)
```

Syncs the HW memory cache for the CPU. Valid only for memory types NVBUF_MEM_SURFACE_ARRAY and NVBUF_MEM_HANDLE.

Parameters:

* surf [in] - A pointer to an \ref NvBufSurface structure.

* index [in] - Index of the buffer in the batch. -1 refers to all buffers in the batch.

* plane [in] - Index of a plane in the buffer. -1 refers to all planes in the buffer.

Returns 0 for success, -1 for failure.

API to sync the hardware memory cache for the device

nvbuf_utils:

```
int
NvBufferMemSyncForDevice (int dmabuf_fd, unsigned int plane, void **pVirtAddr)
```

Syncs the hardware memory cache for the device. Refer NvBufferMemMap for the purpose of the function

* dmabuf_fd [in] - DMABUF FD of buffer.

* plane [in] - video frame plane.

* pVirtAddr [in] - Virtual Address pointer of the memory-mapped plane.

Returns 0 for success, -1 for failure.

⇨ **Above nvbuf_utils API to sync the hardware memory cache for device access can be migrated using below nvutils API.**

nvutils:

```
int
NvBufSurfaceSyncForDevice (NvBufSurface *surf, int index, int plane)
```

Syncs the hardware memory cache for the device. Valid only for memory types NVBUF_MEM_SURFACE_ARRAY and NVBUF_MEM_HANDLE.

Parameters:

* surf [in] - A pointer to an \ref NvBufSurface structure.

* index [in] - Index of a buffer in the batch. -1 refers to all buffers in the batch.

* plane [in] - Index of a plane in the buffer. -1 refers to all planes in the buffer.

Returns 0 for success, -1 for failure.

API to get the memory-mapped virtual address of the plane

nvbuf_utils:

```
int
NvBufferMemMap (int dmabuf_fd, unsigned int plane, NvBufferMemFlags memflag, void
**pVirtAddr)
```

Gets the memory-mapped virtual address of the plane. The client must call NvBufferMemSyncForCpu() with the virtual address returned by this function before accessing the mapped memory in CPU. After memory mapping is complete, mapped memory modification must be coordinated between the CPU and hardware device as follows:

- CPU: If the CPU modifies any mapped memory, the client must call NvBufferMemSyncForDevice() before any hardware device accesses the memory.

- HARDWARE DEVICE: If the mapped memory is modified by any hardware device, the client must call NvBufferMemSyncForCpu() before CPU accesses the memory.

Parameters:

* dmabuf_fd [in] - DMABUF FD of buffer.

* plane [in] - video frame plane. (Applies to NvBufferPayload_SurfArray.)

* memflag [in] - NvBuffer memory flag

* pVirtAddr [out] - Virtual Address pointer of the memory-mapped plane.

Returns 0 for success, -1 for failure.

⇨ Above nvbuf_utils API to get plane wise memory mapped virtual address can be migrated using below nvutils API.

nvutils:

```
int
NvBufSurfaceMap (NvBufSurface *surf, int index, int plane, NvBufSurfaceMemMapFlags type)
```

Maps hardware batched buffers to the HOST or CPU address space. Valid for NVBUF_MEM_CUDA_UNIFIED type memory for dGPU and NVBUF_MEM_SURFACE_ARRAY and NVBUF_MEM_HANDLE type memory for Jetson. This function fills an array of pointers at

`surf->surfaceList->mappedAddr->addr`, where

`surf` is a pointer to an NvBufSurface.

`surfaceList` is a pointer to an NvBufSurfaceParams.

`mappedAddr` is a pointer to an NvBufSurfaceMappedAddr.

`addr` is declared as an array of pointers to void and holds pointers to the buffers.

The client must call NvBufferMemSyncForCpu() with the virtual address populated by this function before accessing mapped memory in the CPU. After memory mapping is complete, mapped memory modification must be coordinated between the CPU and the hardware device as follows:

- CPU: If the CPU modifies mapped memory, the client must call NvBufferMemSyncForDevice() before any hardware device accesses the memory.

- HARDWARE DEVICE: If a hardware device modifies mapped memory, the client must call NvBufferMemSyncForCpu() before the CPU accesses the memory.

Use NvBufSurfaceUnMap() to unmap buffer(s) and release any resource.

Parameters:

* surf [in, out] - A pointer to an NvBufSurface structure. The function stores pointers to the buffers in a descendant of this structure; see the notes above.

* index [in] - Index of a buffer in the batch. -1 refers to all buffers in the batch.

* plane [in] - Index of a plane in buffer. -1 refers to all planes in the buffer.

* type [in] - A flag for mapping type.

Returns 0 for success, -1 for failure.

API to unmap the previously mapped virtual address of the plane

nvbuf_utils:

```
int
NvBufferMemUnMap (int dmabuf_fd, unsigned int plane, void **pVirtAddr)
```

Unmaps the mapped virtual address of the plane. If the following conditions are both true, the client must call NvBufferMemSyncForDevice() before unmapping the memory:

- Mapped memory was modified by the CPU.

- Mapped memory will be accessed by a hardware device.

Parameters:

* dmabuf_fd [in] - DMABUF FD of the buffer.

* plane [in] - Video frame plane. Applies to NvBufferPayload_SurfArray.

* pVirtAddr [in] - Virtual address pointer to the memory-mapped plane.

Returns 0 for success, -1 for failure.

⇨ **Above nvbuf_utils API to unmap plane wise previously mapped virtual address can be migrated using below nvutils API.**

nvutils:

```
int
NvBufSurfaceUnMap (NvBufSurface *surf, int index, int plane)
```

Unmaps previously mapped buffer(s).

Parameters

* surf [in] - A pointer to an NvBufSurface structure.

* index [in] - Index of a buffer in the batch. -1 indicates all buffers in the batch.

* plane [in] - Index of a plane in the buffer. -1 indicates all planes in the buffer.

Returns 0 for success, -1 for failure.

API to copy the hardware buffer plane contents to a raw buffer plane

nvbuf_utils:

```
int
NvBuffer2Raw (int dmabuf_fd, unsigned int plane, unsigned int out_width, unsigned int
out_height,
                    unsigned char *ptr)
```

Copies the NvBuffer plane contents to a raw buffer plane.

Parameters:

* dmabuf_fd [in] - DMABUF FD of NvBuffer.

* plane [in] - video frame plane.

* out_width [in] - aligned width of the raw data plane.

* out_height [in] -aligned height of the raw data plane.

* ptr [in] - pointer to the output raw plane data.

Returns 0 for success, -1 for failure.

⇨ **Above nvbuf_utils API to plane wise copy hardware buffer contents to a raw buffer can be migrated using below nvutils API.**

nvutils:

```
int
NvBufSurface2Raw (NvBufSurface *Surf, unsigned int index, unsigned int plane, unsigned int outwidth,
                                    unsigned int outheight, unsigned char *ptr)
```

Copies the NvBufSurface plane memory content to a raw buffer plane for a specific batched buffer. This function can be used to copy plane memory content from source raw buffer pointer to specific destination batch buffer of supported memory type.

Parameters:

* surf [in] - Pointer to NvBufSurface structure.

* index [in] - Index of buffer in the batch.

* plane [in] - Index of plane in buffer.

* out_width [in] - Aligned width of the raw data plane.

* out_height [in] - Aligned height of the raw data plane.

* ptr [in] - Pointer to the output raw plane data.

Returns 0 for success, -1 for failure.

API to copy raw buffer plane contents to the hardware buffer plane

nvbuf_utils:

```
int
Raw2NvBuffer (unsigned char *ptr, unsigned int plane, unsigned int in_width, unsigned int in_height,
                                int dmabuf_fd)
```

Copies raw buffer plane contents to an NvBuffer plane.
Parameters:
* ptr [in] - Pointer to the input raw plane data.
* plane [in] - Video frame plane.
* in_width [in] - Aligned width of the raw data plane.
* in_height [in] - Aligned height of the raw data plane.
* dmabuf_fd [in] - DMABUF FD of NvBuffer.

Returns 0 for success, -1 for failure.

⇨ Above nvbuf_utils API to plan wise copy raw buffer content to hardware buffer can be migrated using below nvutils API.

nvutils:

```
int
Raw2NvBufSurface (unsigned char *ptr, unsigned int index, unsigned int plane, unsigned int in_width,
                                unsigned int in_height, NvBufSurface *surf)
```

Copies the raw buffer plane memory content to the NvBufSurface plane memory of a specific batched buffer. This function can be used to copy plane memory content from batch buffer to specific destination raw buffer pointer.

Parameters:

* ptr [in] - Pointer to the input raw plane data.
* index [in] - Index of buffer in the batch.
* plane [in] - Index of plane in buffer.

* in_width [in] - Aligned width of the raw data plane.

* in_height [in] - Aligned height of the raw data plane.

* surf [in] - Pointer to NvBufSurface structure.

Returns 0 for success, -1 for failure.

API to create an instance of EGLImage from the hardware buffer handle

nvbuf_utils:

EGLImageKHR NvEGLImageFromFd (EGLDisplay display, int dmabuf_fd)

Creates an instance of EGLImage from a DMABUF FD

Parameters:

* display [in] - An EGLDisplay object used during the creation of the EGLImage. If NULL, nvbuf_utils() uses its own instance of EGLDisplay.

* dmabuf_fd [in] - DMABUF FD of the buffer from which the EGLImage is to be created.

Returns EGLImageKHR for success, NULL for failure.

⇨ **Above nvbuf_utils API to create EGLImage instance from hardware buffer handle can be migrated using below nvutils API.**

nvutils:

int NvBufSurfaceMapEglImage (NvBufSurface *surf, int index)

Creates an EGLImage from the memory of one or more NvBufSurface buffers. Only memory type NVBUF_MEM_SURFACE_ARRAY is supported.

This function returns the created EGLImage by storing its address at `surf->surfaceList->mappedAddr->eglImage`, where:

`surf` is a pointer to an NvBufSurface.

`surfaceList` is a pointer to an NvBufSurfaceParams.

`mappedAddr` is a pointer to an NvBufSurfaceMappedAddr.

`eglImage` is declared as a pointer to void and holds an EGLImageKHR.

You can use this function in scenarios where a CUDA operation on Jetson hardware memory (identified by NVBUF_MEM_SURFACE_ARRAY) is required. The EGLImageKHR struct provided by this function can then be registered with CUDA for further CUDA operations.

Parameters:

* surf [in,out] - A pointer to an NvBufSurface structure. The function stores a pointer to the created EGLImage in a descendant of this structure; see the notes above.

* index [in] - Index of a buffer in the batch. -1 specifies all buffers in the batch.

Returns 0 for success, -1 for failure.

API to destroy EGLImage object from the hardware buffer handle

nvbuf_utils:

```
int
NvDestroyEGLImage (EGLDisplay display, EGLImageKHR eglImage)
```

Destroys an EGLImage object.

Parameters:

*display [in] - An EGLDisplay object used to destroy the EGLImage. If NULL, nvbuf_utils() uses its own instance of EGLDisplay.

*eglImage [in] - The EGLImageKHR object to be destroyed.

Returns 0 for success, -1 for failure.

- ⇨ Above nvbuf_utils API to destroy previously created EGLImage handle from hardware buffer can be migrated using below nvutils API.

nvutils:

```
int
NvBufSurfaceUnMapEglImage (NvBufSurface *surf, int index)
```

Destroys the previously created EGLImage object(s).

Parameters:

* surf [in] - A pointer to an NvBufSurface structure.

* index [in] - The index of a buffer in the batch. -1 specifies all buffers in the batch.

Returns 0 for success, -1 for failure.

API to perform synchronous transforms on the input hardware buffer

nvbuf_utils:

```
int
NvBufferTransform (int src_dmabuf_fd, int dst_dmabuf_fd, NvBufferTransformParams
                   *transform_params)
```

Transforms one DMA buffer to another DMA buffer. This function can support transforms for copying, scaling, fliping, rotating, and cropping.

Parameters:

* src_dmabuf_fd [in] - DMABUF FD of source buffer

* dst_dmabuf_fd [in] - DMABUF FD of destination buffer

* transform_params [in] - Transform parameters

Returns 0 for success, -1 for failure.

- ⇨ **Above nvbuf_utils API for synchronous buffer transform can be migrated using below nvutils API.**

nvutils:

```
NvBufSurfTransform_Error
NvBufSurfTransform (NvBufSurface *src, NvBufSurface *dst, NvBufSurfTransformParams
                    *transform_params)
```

Performs a transformation on batched input images. If user-defined session parameters are to be used, call NvBufSurfTransformSetSessionParams() before calling this function.

Parameters:

* src [in] - A pointer to input batched buffers to be transformed.

* dst [out] - A pointer to a caller-allocated location where transformed output is to be stored. @par When destination cropping is performed, memory outside the crop location is not touched, and may contain stale information. The caller must perform a memset before calling this function if stale information must be eliminated.

* transform_params [in] - A pointer to an NvBufSurfTransformParams structure which specifies the type of transform to be performed. They may include any combination of scaling, format conversion, and cropping for both source and destination. Flipping and rotation are supported on VIC.

Returns NvBufSurfTransform_Error value indicating success or failure.

API to perform asynchronous (non-blocking) transforms on the input hardware buffer

nvbuf_utils:

```
int
NvBufferTransformAsync (int src_dmabuf_fd, int dst_dmabuf_fd, NvBufferTransformParams
                        *transform_params, NvBufferSyncObj *syncobj)
```

Transforms one DMA buffer to another DMA buffer asynchronously (non-blocking). This function can support transforms for copying, scaling, flipping, rotating, and cropping.

Parameters:

* src_dmabuf_fd [in] - DMABUF FD of source buffer

* dst_dmabuf_fd [in] - DMABUF FD of destination buffer

* transform_params [in] - Transform parameters

* syncobj [in] - Nvbuffer sync point object

Returns 0 for success, -1 for failure.

⇨ Above nvbuf_utils API for asynchronous(non-blocking) buffer transform can be migrated using below nvutils API.

nvutils:

```
NvBufSurfTransform_Error

NvBufSurfTransformAsync (NvBufSurface *src, NvBufSurface *dst,
NvBufSurfTransformParams
                                    *transform_params, NvBufSurfTransformSyncObj_t *sync_obj)
```

An asynchronous (non-blocking) transformation on batched input images. If user-defined session parameters are to be used, call NvBufSurfTransformSetSessionParams() before calling this function.

Parameters:

* src [in] - A pointer to input batched buffers to be transformed.

* dst [out] - A pointer to a caller-allocated location where transformed output is to be stored.

When destination cropping is performed, memory outside the crop location is not touched, and may contain stale information. The caller must perform a memset before calling this function if stale information must be eliminated.

* transform_params [in] - A pointer to an NvBufSurfTransformParams structure which specifies the type of transform to be performed. They may include any combination of scaling, format conversion, and cropping for both source and destination. Flipping and rotation are supported on VIC/GPU.

* sync_objs [out] - A pointer to an NvBufSurfTransformSyncObj structure which holds synchronization information of the current transform call. NvBufSurfTransfromSyncObjWait() API to be called on this object to wait for transformation to complete. NvBufSurfTransformSyncObjDestroy API should be called after NvBufSurfTransformSyncObjWait API to release the objects If the parameter is NULL, the call would return only after the transform is complete.

Returns NvBufSurfTransform_Error value indicating success or failure.

API to perform synchronous composition operation on the input hardware buffers

nvbuf_utils:

```
int
NvBufferComposite (int *src_dmabuf_fds, int dst_dmabuf_fd, NvBufferCompositeParams
                   *composite_params)
```

This function composites multiple input DMA buffers to one output DMA buffer.

Parameters:

* src_dmabuf_fds [in] - An array of DMABUF FDs of source buffers. These buffers are composited together. Output is copied to the output buffer referenced by dst_dmabuf_fd.

* dst_dmabuf_fd [in] - DMABUF FD of the compositing destination buffer.

* composite_params [in] - Compositing parameters.

Returns 0 for success, -1 for failure.

⇨ **Above nvbuf_utils API to synchronous buffer composition can be migrated using below nvutils API.**

nvutils:

```
NvBufSurfTransform_Error
NvBufSurfTransformMultiInputBufComposite (NvBufSurface **src, NvBufSurface *dst,
                                          NvBufSurfTransformCompositeParams
*composite_params)
```

Performs Composition on multiple input images with single batch size.

The API composites batched (batch size=1) input buffers pointed by src pointer. Compositer scales and stitches batched buffers pointed by src into single dst buffer (batch size=1). The parameters for location to be composited is provided by composite_params. Use NvBufSurfTransformSetSessionParams before each call, if user defined session parameters are to be used.

Parameters:

* src [in] - (Multiple buffers) Double pointer to input batched (batch size=1) buffers to be transformed.

* dst [out] - (Single buffer) Pointer where composited output would be stored.

* composite_params [in] - Pointer to NvBufSurfTransformCompositeParams structure.

Returns NvBufSurfTransform_Error value indicating success or failure.

API to Create/Destroy new NvBufferSession

nvbuf_utils:

NvBufferSession NvBufferSessionCreate (void)

Creates a new NvBufferSession for parallel scheduling of buffer transformations and compositions.

Returns a session pointer and NULL for failure.

void NvBufferSessionDestroy (NvBufferSession session)

Destroys an existing NvBufferSession.

Parameters:

* session [in] – An existing NvBufferSession.

⇨ **Above nvbuf_utils APIs do not have corresponding nvutils interface APIs. Nvutils provides below interface APIs instead to Set/Get user-defined session parameters for the default session used by Transform and Composite related APIs of nvbufsurftransform.h**

nvutils:

NvBufSurfTransform_Error

NvBufSurfTransformSetSessionParams (NvBufSurfTransformConfigParams *config_params)

Performs Composition on multiple input images with single batch size. The API sets user-defined session parameters. If user-defined session parameters are set, they override the NvBufSurfTransform() function's default session.

Parameters:

* config_params [in] – A pointer to a structure that is populated with the session parameters to be used.

Returns NvBufSurfTransform_Error value indicating success or failure.

NvBufSurfTransform_Error

NvBufSurfTransformGetSessionParams (NvBufSurfTransformConfigParams *config_params)

Gets the session parameters used by NvBufSurfTransform()

Parameters:

* config_params [out] – A pointer to a caller-allocated structure to be populated with the session parameters used.

Returns NvBufSurfTransform_Error value indicating success or failure.

API to get hardware buffer structure size

nvbuf_utils:

int

NvBufferGetSize (void)

This method is used to get hardware buffer struct size.

Returns hardware buffer struct size.

- Above nvbuf_utils API do not have corresponding nvutils interface API. Hardware buffer struct size can be retrieved from NvBufSurface struct itself for Nvidia internal usage only.

Notice

Trademarks

Copyright