`

ACCELERATED GSTREAMER USER GUIDE

DA_07303-4.0 | July 2, 2019

Release 32.2

DOCUMENT CHANGE HISTORY

DA_07303-4.0

Version	Date	Authors	Description of Change
v1.0	01 May 2015	NVIDIA	Initial release.
v1.1	30 Jun 2015	mzensius	Added rotation and scaling commands, other new content.
v1.2	03 Nov 2015	emilyh	Changes for 23.1
v1.3	19 Nov 2015	mzensius	Added note for display export.
v1.4	17 Dec 2015	hlang	Updated gst-nvivafilter sample pipelines. Updated steps to build gstreamer manually.
v1.5	08 Jan 2016	kstone	Added nvvidconv interpolation method.
v1.5	29 Jan 2016	hlang	Additional syntax changes for 23.2 release
v2.0	11 May 2016	mzensius	Minor change to nvgstcapture options.
v3.0	11 Aug 2016	mzensius	Versioned for 24.2 release. GStreamer-0.10 content removed. Also Adds Video Cropping example, interpolation methods for video scaling, EGLStream producer example, and an EGL Image transform example.
v3.1	06 Oct 2016	mzensius	Minor updates to video encoder features.
v3.1.1	21 Nov 2016	mzensius	Changed title of document.
V3.2	12 Jan 2017	mzensius	Adds H.264/H.265 encoder documentation. Also corrects the GStreamer-1.0 installation procedure.
3.2	03 Mar 2017	hlang	Update date/moniker for L4T 27.1 release. No other updates.
3.3	13 Jul 2017	mzensius	Minor edit to command syntax, and update of date/moniker for L4T 28.1 release.
3.4	01 Dec 2017	mzensius	Includes support for NVIDIA® Jetson™ TX1, previously documented elsewhere. Also includes Overlay Sink information, and formatting enhancements.
3.5	23 Feb 2018	kstone	Added support for the nvarguscamerasrc plugin. Corrected erroneous path. Reformatted commands for line breaks.
3.5	28 Feb 2018	hlang	Update the GStreamer installation and setup table to add nvcompositor.

3.6	20 April 2018	kstone	Added prerequisites for Video Composition.
3.7	23 July 2018	jsachs	Add steps to be performed when testing Wayland based GST plugin, playing video, or running Wayland based apps on Wayland display server.
3.8	29 August 2018	jsachs	Updates for L4T release 31 & GStreamer version 1.14.
3.9	2 November 2018	Jsachs	Updates for L4T release 31.1. Jetson TX1 and TX2 not supported.
4.0	2 July 2019	jsachs	Updates for L4T release 32.1. Jetson™ TX2 and Jetson Nano™ supported.

TABLE OF CONTENTS

ACCELERATED GSTREAMER USER GUIDE

This document is a user guide for the GStreamer version 1.0 based accelerated solution included in NVIDIA® Tegra® Linux Driver Package (**L4T**) for NVIDIA® Jetson™ Nano, NVIDIA® Jetson AGX Xavier™, and NVIDIA® Jetson™ TX2 series devices.

Note **References to GStreamer version 1.0 apply to GStreamer version 1.14.**

GSTREAMER-1.0 INSTALLATION AND SETUP

This section describes how to install and configure GStreamer.

To install GStreamer-1.0

- Install GStreamer-1.0 on the platform with the following commands:

```
sudo add-apt-repository universe
sudo add-apt-repository multiverse
sudo apt-get update
sudo apt-get install gstreamer1.0-tools gstreamer1.0-alsa \
  gstreamer1.0-plugins-base gstreamer1.0-plugins-good \
  gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly \
  gstreamer1.0-libav
sudo apt-get install libgstreamer1.0-dev \
  libgstreamer-plugins-base1.0-dev \
  libgstreamer-plugins-good1.0-dev \
  libgstreamer-plugins-bad1.0-dev
```

To check the GStreamer-1.0 version

- Check the GStreamer-1.0 version with the following command:

```
gst-inspect-1.0 --version
```

GStreamer-1.0 Plugin Reference

Note **The `gst-omx` plugin is deprecated in Linux for Tegra (L4T) Release 32.1. Use the `gst-v4l2` plugin instead.**

GStreamer version 1.0 includes the following `gst-omx` video decoders:

Video Decoder	Description
omxh265dec	OpenMAX IL H.265 Video decoder
omxh264dec	OpenMAX IL H.264 Video decoder
omxmpeg4videodec	OpenMAX IL MPEG4 Video decoder
omxmpeg2videodec	OpenMAX IL MPEG2 Video decoder
omxvp8dec	OpenMAX IL VP8 Video decoder
omxvp9dec	OpenMAX IL VP9 video decoder

GStreamer version 1.0 includes the following gst-v4l2 video decoders:

Video Encoders	Description
nvv4l2decoder	V4L2 H.265 Video decoder
	V4L2 H.264 Video decoder
	V4L2 VP8 video decoder
	V4L2 VP9 video decoder
	V4L2 MPEG4 video decoder
	V4L2 MPEG2 video decoder

GStreamer version 1.0 includes the following gst-omx video encoders:

Video Encoders	Description
omxh264enc	OpenMAX IL H.264/AVC video encoder
omxh265enc	OpenMAX IL H.265/AVC video encoder
omxvp8enc	OpenMAX IL VP8 video encoder (supported with NVIDIA® Jetson™ TX2/TX2i and NVIDIA® Jetson Nano™; not supported with NVIDIA® Jetson AGX Xavier™)
omxvp9enc	OpenMAX IL VP9 video encoder (supported with Jetson TX2 and Jetson AGX Xavier; not supported with Jetson Nano)

GStreamer version 1.0 includes the following gst-v4l2 video encoders:

Video Encoders	Description
nvv4l2h264enc	V4l2 H.264 video encoder
nvv4l2h265enc	V4l2 H.265 video encoder
nvv4l2vp8enc	V4l2 VP8 video encoder (supported with Jetson TX2/TX2i and Jetson Nano; not supported with Jetson AGX Xavier)
nvv4l2vp9enc	V4l2 VP9 video encoder (supported with Jetson AGX Xavier and Jetson; not supported with Jetson Nano)

GStreamer version 1.0 includes the following gst-omx video sink:

Video Sink	Description
nvoverlaysink	OpenMAX IL videosink element

GStreamer version 1.0 includes the following EGL image video sink:

Video Sink	Description
nveglglessink	EGL/GLES videosink element, both the X11 and Wayland backends
nv3dsink	EGL/GLES videosink element

GStreamer version 1.0 includes the following DRM video sink:

Video Sink	Description
nvdrmvideosink	DRM videosink element

Note **The `nvoverlaysink` plugin is deprecated in L4T Release 32.1. Use `nvdrmvideosink` and `nv3dsink` instead for render pipelines with `gst-v4l2` decoder.**

GStreamer version 1.0 includes the following proprietary NVIDIA plugins:

NVIDIA Proprietary Plugin	Description
nvarguscamerasrc	Camera plugin for ARGUS API
nvvidconv	Video format conversion & scaling
nvcompositor	Video compositor
nveglstreamsrc	Acts as GStreamer Source Component, accepts EGLStream from EGLStream producer
nvvideosink	Video Sink Component. Accepts YUV-I420 format and produces EGLStream (RGBA)
nvegltransform	Video transform element for NVMM to EGLimage (supported with nveglglessink only)

GStreamer version 1.0 includes the following libjpeg based JPEG image video encode/decode plugins:

JPEG	Description
nvjpegenc	JPEG encoder element
nvjpegdec	JPEG decoder element

Note **Execute this command on the target before starting the video decode pipeline using gst-launch or nvgstplayer.**

export DISPLAY=:0

Start the X server with xinit &, if it is not already running.

DECODE EXAMPLES

The examples in this section show how you can perform audio and video decode with GStreamer.

Note **GStreamer version 0.10 support is deprecated in Linux for Tegra (L4T) Release 24.2. Use of GStreamer version 1.0 is recommended for development.**

Audio Decode Examples Using gst-launch-1.0

The following examples show how you can perform audio decode using GStreamer-1.0.

AAC Decode (OSS Software Decode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.audio_0 ! \
  queue ! avdec_aac ! audioconvert ! alsasink -e
```

AMR-WB Decode (OSS Software Decode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.audio_0 ! queue ! avdec_amrwb ! \
  audioconvert ! alsasink -e
```

AMR-NB Decode (OSS Software Decode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.audio_0 ! queue ! avdec_amrnb ! \
  audioconvert ! alsasink -e
```

MP3 Decode (OSS Software Decode)

```
gst-launch-1.0 filesrc location=<filename.mp3> ! mpegaudioparse ! \
  avdec_mp3 ! audioconvert ! alsasink -e
```

Note **To route audio over HDMI, set the alsasink property device as follows:**

`hw:Tegra,3`

Video Decode Examples Using gst-launch-1.0

The following examples show how you can perform video decode on GStreamer-1.0.

Video Decode Using gst-omx

The following examples show how you can perform video decode using the gst-omx plugin on GStreamer-1.0.

H.264 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h264parse ! omxh264dec ! \
  nveglglessink -e
```

H.265 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h265parse ! omxh265dec ! \
  nvoverlaysink -e
```

10-bit H.265 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_10bit.mkv> ! \
  matroskademux ! h265parse ! omxh265dec ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)NV12' ! \
  nvoverlaysink -e
```

12-bit H.265 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_12bit.mkv> ! \
  matroskademux ! h265parse ! omxh265dec ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)NV12' ! \
  nvoverlaysink -e
```

Note **For decode use cases with low memory allocation requirements (e.g. on Jetson Nano), use the `enable-low-outbuffer` property of the `gst-omx` decoder plugin.**

For an example, see the pipeline below.

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux ! h265parse ! omxh265dec enable-low-outbuffer=1 ! \
'video/x- raw(memory:NVMM), format=(string)NV12' ! fakesink sync=1 -e
```

Note **To enable max perf mode, use the `disable-dvfs` property of the `gst-omx` decoder plugin. Expect increased power consumption in max perf mode.**

For an example, see the pipeline below.

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux ! h265parse ! omxh265dec disable-dvfs=1 ! \
'video/x-raw(memory:NVMM), format=(string)NV12' ! fakesink sync=1 -e
```

VP8 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! omxvp8dec ! \
  nvoverlaysink -e
```

Note **If the primary display is NOT used to render video, use the `display-id` property of `nvoverlaysink`.**

For example, refer to the pipeline below.

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! omxvp8dec ! \
  nvoverlaysink display-id=1 -e
```

VP9 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  matroskademux name=demux demux.video_0 ! queue ! omxvp9dec ! \
  nvoverlaysink display-id=1 -e
```

MPEG-4 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! mpeg4videoparse ! \
  omxmpeg4videodec ! nveglglessink -e
```

MPEG-2 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename.ts> ! \
  tsdemux name=demux demux.video_0 ! queue ! mpegvideoparse ! \
  omxmpeg2videodec ! nveglglessink -e
```

Video Decode Using gst-v4l2

The following examples show how you can perform video decode using gst-v4l2 plugin on GStreamer-1.0.

H.264 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_h264.mp4> ! \
  qtdemux ! queue ! h264parse ! nvv4l2decoder ! nv3dsink -e
```

Note

To enable max perf mode, use the `enable-max-performance` property of the `gst-v4l2` decoder plugin. Expect increased power consumption in max perf mode.

For an example, see the pipeline below.

```
gst-launch-1.0 filesrc location=<filename_h264.mp4> ! \
  qtdemux ! queue ! h264parse ! nvv4l2decoder \
  enable-max-performance=1 ! nv3dsink -e
```

H.265 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_h265.mp4> ! \
  qtdemux ! queue ! h265parse ! nvv4l2decoder ! nv3dsink -e
```

10-bit H.265 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_10bit.mkv> ! \
  matroskademux ! queue ! h265parse ! nvv4l2decoder ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)NV12' ! nv3dsink -e
```

12-bit H.265 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_12bit.mkv> ! \
  matroskademux ! queue ! h265parse ! nvv4l2decoder ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)NV12' ! nv3dsink -e
```

VP9 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_vp9.mkv> ! \
  matroskademux ! queue ! nvv4l2decoder ! nv3dsink -e
```

VP8 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_vp8.mkv> ! \
  matroskademux ! queue ! nvv4l2decoder ! nv3dsink -e
```

MPEG-4 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_mpeg4.mp4> ! \
  qtdemux ! queue ! mpeg4videoparse ! nvv4l2decoder ! nv3dsink -e
```

MPEG-4 Decode DivX 4/5 (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_divx.avi> ! \
  avidemux ! queue ! mpeg4videoparse ! nvv4l2decoder ! nv3dsink -e
```

MPEG-2 Decode (NVIDIA Accelerated Decode)

```
gst-launch-1.0 filesrc location=<filename_mpeg2.ts> ! \
  tsdemux ! queue ! mpegvideoparse ! nvv4l2decoder ! nv3dsink -e
```

Image Decode Examples Using gst-launch-1.0

The following examples show how you can perform JPEG decode on GStreamer-1.0.

```
gst-launch-1.0 filesrc location=<filename.jpg> ! nvjpegdec ! \
  imagefreeze ! xvimagesink -e
```

ENCODE EXAMPLES

The examples in this section show how you can perform audio and video encode with GStreamer.

Audio Encode Examples Using gst-launch-1.0

The following examples show how you can perform audio encode on GStreamer-1.0.

AAC Encode (OSS Software Encode)

```
gst-launch-1.0 audiotestsrc ! \
  'audio/x-raw, format=(string)S16LE,
  layout=(string)interleaved, rate=(int)44100, channels=(int)2' ! \
  voaacenc ! qtmux ! filesink location=test.mp4 -e
```

AMR-WB Encode (OSS Software Encode)

```
gst-launch-1.0 audiotestsrc ! \
  'audio/x-raw, format=(string)S16LE, layout=(string)interleaved, \
  rate=(int)16000, channels=(int)1' ! voamrwbenc ! qtmux ! \
  filesink location=test.mp4 -e
```

Video Encode Examples Using gst-launch-1.0

The following examples show how you can perform video encode with GStreamer-1.0.

Video Encode Using gst-omx

The following examples show how you can perform video encode using the gst-omx plugin with GStreamer-1.0.

H.264 Encode (NVIDIA Accelerated Encode)

```
gst-launch-1.0 videotestsrc ! \
  'video/x-raw, format=(string)I420, width=(int)640, \
  height=(int)480' ! omxh264enc ! \
  'video/x-h264, stream-format=(string)byte-stream' ! h264parse ! \
  qtmux ! filesink location=test.mp4 -e
```

H.265 Encode (NVIDIA Accelerated Encode)

```
gst-launch-1.0 videotestsrc ! \
  'video/x-raw, format=(string)I420, width=(int)640, \
  height=(int)480' ! omxh265enc ! filesink location=test.h265 -e
```

10-bit H.265 Encode (NVIDIA Accelerated Encode)

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! \
  nvvidconv ! 'video/x-raw(memory:NVMM), format=(string)I420_10LE' !
  omxh265enc ! matroskamux ! filesink location=test_10bit.mkv -e
```

VP8 Encode (NVIDIA Accelerated, Supported with Jetson TX2/TX2i and Jetson Nano)

```
gst-launch-1.0 videotestsrc ! \
  'video/x-raw, format=(string)I420, width=(int)640, \
  height=(int)480' ! omxvp8enc ! matroskamux ! \
  filesink location=test.mkv -e
```

VP9 Encode (NVIDIA Accelerated, Supported with Jetson TX2 and Jetson AGX Xavier)

```
gst-launch-1.0 videotestsrc ! \
  'video/x-raw, format=(string)I420, width=(int)640, \
  height=(int)480' ! omxvp9enc ! matroskamux ! \
  filesink location=test.mkv -e
```

MPEG-4 Encode (OSS Software Encode)

```
gst-launch-1.0 videotestsrc ! \
  'video/x-raw, format=(string)I420, width=(int)640, \
  height=(int)480' ! avenc_mpeg4 ! qtmux ! \
  filesink location=test.mp4 -e
```

H.263 Encode (OSS Software Encode)

```
gst-launch-1.0 videotestsrc ! \
  'video/x-raw, format=(string)I420, width=(int)704, \
  height=(int)576' ! avenc_h263 ! qtmux ! filesink location=test.mp4 -e
```

Video Encode Using gst-v4l2

The following examples show how you can perform video encode using gst-v4l2 plugin with GStreamer-1.0.

H.264 Encode (NVIDIA Accelerated Encode)

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvv4l2h264enc ! \
  bitrate-8000000 ! h264parse ! qtmux ! filesink \
  location=<filename_h264.mp4> -e
```

Note

To enable max perf mode, use the `maxperf-enable` property of the `gst-v4l2` encoder plugin. Expect increased power consumption in max perf mode.

For an example, see the pipeline below.

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvv4l2h264enc \
  maxperf-enable=1 bitrate=8000000 ! h264parse ! qtmux ! filesink \
  location=<filename_h264.mp4> -e
```

H.265 Encode (NVIDIA Accelerated Encode)

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvv4l2h265enc \
  bitrate=8000000 ! h265parse ! qtmux ! filesink \
  location=<filename_h265.mp4> -e
```

10-bit H.265 Encode (NVIDIA Accelerated Encode)

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)P010_10LE' ! \
  nvv4l2h265enc bitrate=8000000 ! h265parse ! qtmux ! \
  filesink location=<filename_10bit_h265.mp4> -e
```

VP9 Encode (NVIDIA Accelerated Encode)

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvv4l2vp9enc \
  bitrate=8000000 ! matroskamux ! filesink \
  location=<filename_vp9.mkv> -e
```

VP9 Encode with IVF Headers (NVIDIA Accelerated Encode)

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvv4l2vp9enc \
  enable-headers=1 bitrate=8000000 ! filesink \
  location=<filename_vp9.vp9> -e
```

VP8 Encode (NVIDIA Accelerated Encode)

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvv4l2vp8enc \
  bitrate=8000000 ! matroskamux ! filesink \
  location=<filename_vp8.mkv> -e
```

VP8 Encode with IVF Headers (NVIDIA Accelerated Encode)

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvv4l2vp8enc \
  enable-headers=1 bitrate=8000000 ! filesink \
  location=<filename_vp8.vp8> -e
```

Image Encode Examples Using gst-launch-1.0

The following examples show how you can perform JPEG encode on GStreamer-1.0 .

Image Encode

```
gst-launch-1.0 videotestsrc num-buffers=1 ! \
  'video/x-raw, width=(int)640, height=(int)480, \
  format=(string)I420' ! nvjpegenc ! filesink location=test.jpg -e
```

Supported H.264/H.265/VP8/VP9 Encoder Features with GStreamer-1.0

This section describes example gst-launch-1.0 usage for features supported by the NVIDIA accelerated H.264/H.265/VP8/VP9 encoders.

Features Supported Using gst-omx

This section describes example gst-launch-1.0 usage for features supported by the NVIDIA accelerated H.264/H.265/VP8/VP9 `gst-omx` encoders.

Note

Display detailed information on omxh264enc or omxh265enc encoder properties with the `gst-inspect-1.0 [omxh264enc | omxh265enc | omxvp8enc | omxvp9enc]` command.

Set I-Frame Interval

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! omxh264enc iframeinterval=100 ! qtmux ! \
  filesink location=test.mp4 -e
```

Set Temporal-Tradeoff (Rate at Which the Encoder Should Drop Frames)

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! omxh264enc temporal-tradeoff=1 ! qtmux ! \
  filesink location=test.mp4 -e
```

Configuring temporal tradeoff causes the encoder to intentionally, periodically, drop input frames. The following modes are supported:

Mode	Description
0	Disable
1	Drop 1 in 5 frames
2	Drop 1 in 3 frames
3	Drop 1 in 2 frames
4	Drop 2 in 3 frames

Set Rate Control Mode

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! omxh264enc control-rate=1 ! qtmux ! \
  filesink location=test.mp4 -e
```

The following modes are supported:

Mode	Description
0	Disable
1	Variable bit rate
2	Constant bit rate
3	Variable bit rate with frame skip. The encoder skips frames as necessary to meet the target bit rate.
4	Constant bit rate with frame skip

Set Peak Bitrate

```
gst-launch-1.0 videotestsrc num-buffers=200 is-live=true ! \
  'video/x-raw,width=1280,height=720,format=I420' ! \
  omxh264enc bitrate=6000000 peak-bitrate=6500000 ! qtmux ! \
  filesink location=test.mp4 -e
```

It takes effect only in variable bit rate(control-rate=1) mode. By default, the value is configured as (1.2*bitrate).

Set Quantization Range for I, P and B Frame

The format for the range is the following:

```
"<I_range>:<P_range>:<B_range>"
```

Where `<I_range>`, `<P_range>` and `<B_range>` are each expressed as hyphenated values, as shown in the following example:

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! \
  omxh264enc qp-range="10,30:10,35:10,35" ! qtmux ! \
  filesink location=test.mp4 -e
```

The range of B frames does not take effect if the number of B frames is 0.

Set Hardware Preset Level

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! omxh264enc preset-level=0 ! qtmux ! \
  filesink location=test.mp4 -e
```

The following modes are supported:

Mode	Description
0	UltraFastPreset
1	FastPreset Only Integer Pixel (integer-pel) block motion is estimated. For I/P macroblock mode decision, only Intra 16 x 16 cost is compared with Inter modes costs. Supports Intra 16 x 16 and Intra 4 x 4 modes.
2	MediumPreset Supports up to Half Pixel (half-pel) block motion estimation. For an I/P macroblock mode decision, only Intra 16 x 16 cost is compared with Inter modes costs. Supports Intra 16 x 16 and Intra 4 x 4 modes.
3	SlowPreset Supports up to Quarter Pixel (Qpel) block motion estimation. For an I/P macroblock mode decision, Intra 4 x 4 as well as Intra 16 x 16 cost is compared with Inter modes costs. Supports Intra 16 x 16 and Intra 4 x 4 modes.

Set Profile

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! omxh264enc profile=8 ! qtmux ! \
  filesink location=test.mp4 -e
```

From `omxh264enc`, the following profiles are supported:

Profile	Description
1	Baseline profile
2	Main profile
8	High profile

Set Level

```
gst-launch-1.0 videotestsrc num-buffers=200 is-live=true ! \
  'video/x-raw, format=(string)I420, width=(int)256, height=(int)256, \
  framerate=(fraction)30/1' ! omxh264enc bitrate=40000 !\
  'video/x-h264, level=(string)2.2' ! qtmux ! \
  filesink location= test.mp4 -e
```

From omxh264enc, the following levels are supported: `1, 1b, 1.2, 1.3, 2, 2.1, 2.2, 3, 3.1, 3.2, 4, 4.1, 4.2, 5, 5.1,` and `5.2`.

From omxh265enc, the following levels are supported: `main1, main2, main2.1, main3, main3.1, main4, main4.1, main5, high1, high2, high2.1, high3, high3.1, high4, high4.1,` and `high5`.

Set Number of B Frames Between Two Reference Frames

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! omxh264enc num-B-Frames=2 ! qtmux ! \
  filesink location=test.mp4 -e
```

Note **B-frame-encoding is not supported with omxh265enc.**

Insert SPS and PPS at IDR

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! omxh264enc insert-sps-pps=1 ! qtmux ! \
  filesink location=test.mp4 -e
```

If enabled, a sequence parameter set (SPS) and a picture parameter set (PPS) are inserted before each IDR frame in the H.264/H.265 stream.

Enable Two-Pass CBR

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! omxh264enc EnableTwopassCBR=1
  control-rate=2 ! qtmux ! filesink location=test.mp4 -e
```

Two-pass CBR must be enabled along with constant bit rate (control-rate=2).

Set Virtual Buffer Size

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! omxh264enc vbv-size=10 ! qtmux ! \
  filesink location=test.mp4 -e
```

If the buffer size of decoder or network bandwidth is limited, configuring virtual buffer size can cause video stream generation to correspond to the limitations according to the following formula:

```
virtual buffer size = vbv-size * (bitrate/fps)
```

Enable Stringent Bitrate

```
gst-launch-1.0 nvarguscamerasrc num-buffers=200 ! \
  'video/x-raw(memory:NVMM),width=1920,height=1080,
  format=(string)NV12' ! \
  omxh264enc control-rate=2 vbv-size=1 EnableTwopassCBR=true \
  EnableStringentBitrate=true ! qtmux ! filesink location=test.mp4 -e
```

Stringent Bitrate must be enabled along with constant bit rate (control-rate=2), two-pass CBR being enabled, and virtual buffer size being set.

Slice-Header-Spacing with Spacing in Terms of MB

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! \
  omxh264enc slice-header-spacing=200 bit-packetization=0 ! \
  qtmux ! filesink location=test.mp4 -e
```

The parameter `bit-packetization=0` configures the network abstraction layer (NAL) packet as macroblock (MB)-based, and `slice-header-spacing=200` configures each NAL packet as 200 MB at maximum.

Slice Header Spacing with Spacing in Terms of Number of Bits

```
gst-launch-1.0 videotestsrc num-buffers=200 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420' ! \
  omxh264enc slice-header-spacing=1024 bit-packetization=1 ! \
  qtmux ! filesink location=test1.mp4 -e
```

The parameter `bit-packetization=1` configures the network abstraction layer (NAL) packet as size-based, and `slice-header-spacing=1024` configures each NAL packet as 1024 bytes at maximum.

Features Supported Using gst-v4l2

This section describes example gst-launch-1.0 usage for features supported by the NVIDIA accelerated H.264/H.265/VP8/VP9 `gst-v4l2` encoders.

Note **Display detailed information on the nvv4l2h264enc, nvv4l2h265enc, or nvv4l2vp9enc encoder property with the `gst-inspect-1.0 [nvv4l2h264enc | nvv4l2h265enc | nvv4l2vp8enc | nvv4l2vp9enc]` command.**

Set I-Frame Interval (Supported with H.264/H.265/VP9 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  iframeinterval=100 ! h264parse ! qtmux ! filesink \
  location=<filename_h264.mp4> -e
```

This property sets encoding Intra Frame occurrence frequency.

Set Rate Control Mode and Bitrate (Supported with H.264/H.265/VP9 Encode)

- Set Variable Bitrate mode:

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  control-rate=0 bitrate=30000000 ! h264parse ! qtmux ! filesink \
  location=<filename_h264_VBR.mp4> -e
```

- Set Constant Bitrate mode:

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  control-rate=1 bitrate=30000000 ! h264parse ! qtmux ! filesink \
  location=<filename_h264_CBR.mp4> -e
```

The following modes are supported:

Mode	Description
0	Variable bit rate (VBR)
1	Constant bit rate (CBR)

Set Quantization Range for I, P and B frame (Supported with H.264/H.265 Encode)

```
gst-launch-1.0 videotestsrc  num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  ratecontrol-enable=0 quant-i-frames=30 quant-p-frames=30 \
  quant-b-frames=30 num-B-Frames=1 ! filesink \
  location=<filename_h264.264> -e
```

The range of B frames does not take effect if the number of B frames is 0.

Set Hardware Preset Level (Supported with H.264/H.265/VP9 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  preset-level=4 MeasureEncoderLatency=1 ! 'video/x-h264, \
  stream-format=(string)byte-stream, alignment=(string)au' ! \
  filesink location=<filename_h264.264> -e
```

The following modes are supported:

Mode	Description
0	DisablePreset
1	UltraFastPreset
2	FastPreset Only Integer Pixel (integer-pel) block motion is estimated. For I/P macroblock mode decisions, only Intra 16×16 cost is compared with intermode costs. Supports intra 16×16 and intra 4×4 modes.
3	MediumPreset Supports up to Half Pixel (half-pel) block motion estimation. For I/P macroblock mode decisions, only Intra 16×16 cost is compared with intermode costs. Supports intra 16×16 and intra 4×4 modes.
4	SlowPreset Supports up to Quarter Pixel (Qpel) block motion estimation. For I/P macroblock mode decisions, intra

Mode	Description
	4×4 as well as intra 16×16 cost is compared with intermode costs. Supports intra 16×16 and intra 4×4 modes.

Set Profile (Supported with H.264 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  profile=0 ! 'video/x-h264, stream-format=(string)byte-stream, \
  alignment=(string)au' ! filesink location=<filename_h264.264> -e
```

The following profiles are supported:

Profile	Description
0	Baseline profile
2	Main profile
4	High profile

Insert SPS and PPS at IDR (Supported with H.264 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  insert-sps-pps=1 ! \
  'video/x-h264, stream-format=(string)byte-stream, \
  alignment=(string)au' ! filesink location=<filename_h264.264> -e
```

If enabled, a sequence parameter set (SPS) and a picture parameter set (PPS) are inserted before each IDR frame in the H.264 stream.

Enable Two-Pass CBR (Supported with H.264/H.265 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  control-rate=1 bitrate=10000000 EnableTwopassCBR=1 ! \
  'video/x-h264, stream-format=(string)byte-stream, \
  alignment=(string)au' ! filesink location=<filename_h264.264> -e
```

Two-pass CBR must be enabled along with constant bit rate (control-rate=1).

Note **For multi-instance encode with two-pass CBR enabled, enable max perf mode by using the `maxperf-enable` property of the `gst-v4l2` encoder to achieve best performance. Expect increased power consumption in max perf mode.**

Slice-Header-Spacing with Spacing in Terms of MB (Supported with H.264/H.265 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  slice-header-spacing=8 bit-packetization=0 ! 'video/x-h264, \
  stream-format=(string)byte-stream, alignment=(string)au' ! \
  filesink location=<filename_h264.264> -e
```

The parameter `bit-packetization=0` configures the network abstraction layer (NAL) packet as macroblock (MB)-based, and `slice-header-spacing=8` configures each NAL packet as 8 MB at maximum.

Slice Header Spacing with Spacing in Terms of Number of Bits (Supported with H.264/H.265 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  slice-header-spacing=1400 bit-packetization=1 ! 'video/x-h264, \
  stream-format=(string)byte-stream, alignment=(string)au' ! \
  filesink location=<filename_h264.264> -e
```

The parameter `bit-packetization=1` configures the network abstraction layer (NAL) packet as size-based, and `slice-header-spacing=1400` configures each NAL packet as 1400 bytes at maximum.

Enable Cabac-Entropy-Coding (Supported with H.264 Encode for Main or High Profile)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  profile=2 cabac-entropy-coding=1 ! 'video/x-h264, \
  stream-format=(string)byte-stream, alignment=(string)au' ! \
  filesink location=<filename_h264.264> -e
```

The following entropy coding types are supported:

Entropy Coding Type	Description
0	CAVLC
1	CABAC

Set Number of B Frames Between Two Reference Frames (Supported with H.264 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  num-B-Frames=1 ! 'video/x-h264, stream-format=(string)byte-stream, \
  alignment=(string)au' ! filesink location=<filename_h264.264> -e
```

This property sets the number of B frames between two reference frames.

Note **For multi-instance encode with `num-B-Frames=2`, enable max perf mode by specifying the `maxperf-enable` property of the `gst-v4l2` encoder for best performance. Expect increased power consumption in max perf mode.**

Set qp-range (Supported with H.264/H.265 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  qp-range="24,24:28,28:30,30" num-B-Frames=1 ! 'video/x-h264, \
  stream-format=(string)byte-stream, alignment=(string)au' ! filesink \
  location=<filename_h264.264> -e
```

This property sets qunatization range for P, I and B frames.

Enable MVBufferMeta (Supported with H.264/H.265 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  EnableMVBufferMeta=1 ! 'video/x-h264, \
  stream-format=(string)byte-stream, alignment=(string)au' ! \
  filesink location=<filename_h264.264> -e
```

This property enables motion vector metadata for encoding.

Insert AUD (Supported with H.264/H.265 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  insert-aud=1 ! 'video/x-h264, stream-format=(string)byte-stream, \
  alignment=(string)au' ! filesink location=<filename_h264.264> -e
```

This property inserts an H.264/H.265 Access Unit Delimiter (AUD).

Insert VUI (Supported with H.264/H.265 Encode)

```
gst-launch-1.0 videotestsrc num-buffers=300 ! \
  'video/x-raw, width=(int)1280, height=(int)720, \
  format=(string)I420, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvv4l2h264enc \
  insert-vui=1 ! 'video/x-h264, stream-format=(string)byte-stream, \
  alignment=(string)au' ! filesink location=<filename_h264.264> -e
```

This property inserts H.264/H.265 Video Usability Information (VUI) in SPS.

CAMERA CAPTURE WITH GSTREAMER-1.0

For `nvgstcapture-1.0` usage information enter the following command:

```
nvgstcapture-1.0 --help
```

Note **nvgstcapture-1.0 application default only supports ARGUS API using nvarguscamerasrc plugin. Legacy nvcamerasrc plugin support is deprecated.**

For more information, see Nvgstcapture-1.0 Option Reference in this guide.

Capture using nvarguscamerasrc and preview display with overlaysink:

```
gst-launch-1.0 nvarguscamerasrc ! ‘video/x-raw(memory:NVMM), \
  width=(int)1920, height=(int)1080, format=(string)NV12, \
  framerate=(fraction)30/1' ! nvoverlaysink -e
```

Note **For multi-instance encode with `num-B-Frames=2`, enable max perf mode by specifying the `maxperf-enable` property of the `gst-v4l2` encoder for best performance. Expect increased power consumption in max perf mode.**

```
gst-launch-1.0 nvarguscamerasrc maxperf=1 ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvv4l2h265enc \
  control-rate=1 bitrate=8000000 ! 'video/x-h265, \
  stream-format=(string)byte-stream' ! h265parse ! qtmux ! filesink \
  location=test.mp4 -e
```

The `nvgstcapture-1.0` application uses the `v4l2src` plugin to capture still images and video.

The following table shows USB camera support.

USB Camera Support	Feature
YUV	Preview display
	Image capture (VGA, 640 x 480)
	Video capture (480p, 720p, H.264/H.265/VP8/VP9 encode)

raw-yuv Capture (I420 Format) and Preview Display with xvimagesink

```
gst-launch-1.0 v4l2src device="/dev/video0" ! \
  "video/x-raw, width=640, height=480, format=(string)YUY2" ! \
  xvimagesink -e
```

Camera Capture and Encode Support with OpenCV

The OpenCV sample application opencv_nvgstcam simulates the camera capture pipeline. Similarly, the OpenCV sample application opencv_nvgstenc simulates the video encode pipeline.

Both sample applications are based on GStreamer 1.0. They currently are supported only by OpenCV version 3.3.

opencv_nvgstcam: Camera Capture and Preview

To simulate the camera capture pipeline with the opencv_nvgstcam sample application, enter this command:

```
./opencv_nvgstcam --help
```

Note **The `opencv_nvgstcam` application as distributed currently supports only single-instance CSI capture using the d plugin. You can modify and rebuild the application to support GStreamer pipelines for CSI multi-instance capture and USB camera capture using the `v4l2src` plugin. The application uses an OpenCV-based videosink for display.**

For camera CSI capture and preview rendering with OpenCV, enter this command:

```
./opencv_nvgstcam --width=1920 --height=1080 --fps=30
```

opencv_nvgstenc: Camera Capture and Video Encode

To simulate the camera capture and video encode pipeline with the opencv_nvgstenc sample application, enter this command:

```
./opencv_nvgstenc --help
```

Note **The `opencv_nvgstenc` application as distributed currently supports only camera CSI capture using the `nvarguscamerasrc` plugin and video encode in H.264 format using the `nvv4l2h264enc` plugin with an MP4 container file. You can modify and rebuild the application to support GStreamer pipelines for different video encoding formats. The application uses an OpenCV-based videosink for display.**

For camera CSI capture and video encode with OpenCV enter this command:

```
./opencv_nvgstenc --width=1920 --height=1080 --fps=30 --time=60 \
  --filename=test_h264_1080p_30fps.mp4
```

VIDEO PLAYBACK WITH GSTREAMER-1.0

For nvgstplayer-1.0 usage information enter the following command:

```
nvgstplayer-1.0 --help
```

Video can be output to HD displays using the HDMI connector on the platform. The GStreamer-1.0 application supports currently the following video sinks:

Note **The `nvoverlaysink` plugin is deprecated in L4T Release 32.1. Use the `nvdrmvideosink` plugin for development.**

Overlay Sink (Video playback on overlay in full-screen mode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux ! h264parse ! omxh264dec ! nvoverlaysink -e
```

Overlay Sink (Video playback using overlay parameters)

Note: The following steps are required to use the “overlay” property on Jetson-TX2.

1. **Set win_mask with the following commands:**

```
# sudo -s
# cd /sys/class/graphics/fb0
# echo 4 > blank       // Blanks monitor for changing
#                      // display setting.
# echo 0x0 > device/win_mask
#                      // Clears current window setting.
#                      // window setting.
# echo 0x3f > device/win_mask
#                      // Assigns all 6 overlay windows
#                      // in display controller to
#                      // display 0 (fb0).
# echo 0 > blank       // Unblank display.
```

2. **Stop X11 using following command:**

```
$ sudo systemctl stop gdm
$ sudo loginctl terminate-seat seat0
```

For more introduction about the overlay windows in the display controller, please refer to the *TX2 Technical Reference Manual* (TRM).

To use all 6 overlays X11 must be disabled, since it occupies one window. Disabling X11 also helps avoid memory bandwidth contention when using a non X11 overlay.

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4> ! \
  qtdemux ! h264parse ! omxh264dec \
  nvoverlaysink overlay-x=100 overlay-y=100 overlay-w=640 \
  overlay-h=480 overlay=1 \
  overlay-depth=0 & gst-launch-1.0 filesrc \
  location=<filename_1080p.mp4> ! qtdemux ! h264parse ! omxh264dec ! \
  nvoverlaysink overlay-x=250 overlay-y=250 overlay-w=640 \
  overlay-h=480 overlay=2 overlay-depth=1 -e
```

nveglglessink (Windowed video playback, NVIDIA EGL/GLES videosink using default X11 backend)

Use the following command to start the GStreamer pipeline using `nveglglesink` with the default X11 backend:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux ! h264parse ! omxh264dec ! nveglglessink -e
```

This nvgstplayer-1.0 application supports specific window position and dimensions for windowed playback:

```
nvgstplayer-1.0 -i <filename> --window-x=300 --window-y=300 \
  --window-width=500 --window-height=500
```

nveglglessink (Windowed video playback, NVIDIA EGL/GLES videosink using Wayland backend)

You can also use `nveglglsink` with the Wayland backend, instead of the default X11 backend.

Ubuntu 16.04 does not support the Wayland display server. That is, there is no UI support to switch to Wayland from Xorg. You must start the Wayland server (Weston) using the target's shell before performing any Weston based operation.

To start Weston:

The following steps are required before you first run the GStreamer pipeline with the Wayland backend. They are not required on subsequent runs.

1. Stop the display manager:

   ```
   sudo systemctl stop gdm
   sudo loginctl terminate-seat seat0
   ```

2. Unset the `DISPLAY` environment variable:

   ```
   unset DISPLAY
   ```

3. Create a temporary `xdg` directory:

   ```
   mkdir /tmp/xdg
   chmod 700 /tmp/xdg
   ```

4. Start the Weston compositor:

   ```
   sudo XDG_RUNTIME_DIR=/tmp/xdg weston --idle-time=0 &
   ```

To run the GStreamer pipeline with the Wayland backend:

Use the following command to start the GStreamer pipeline using `nveglglesink` with the Wayland backend:

```
sudo XDG_RUNTIME_DIR=/tmp/xdg gst-launch-1.0 filesrc \
  location=<filename.mp4> ! qtdemux name=demux ! h264parse ! \
  omxh264dec ! nveglglessink winsys=wayland
```

DRM Video Sink (Video playback using DRM)

This sink element uses DRM to render video on connected displays.

1. Stop the display manager:

   ```
   sudo systemctl stop gdm
   sudo loginctl terminate-seat seat0
   ```

2. Enter this command to start the GStreamer pipeline using `nvdrmvideosink`:

   ```
   gst-launch-1.0 filesrc location=<filename.mp4> ! \
     qtdemux! queue ! h264parse !nvv4l2decoder ! nvdrmvideosink -e
   ```

Properties

`nvdrmvideosink` supports the following properties:

Property name	Description
conn_id	Set connector ID for display.
plane_id	Set plane ID.
set_mode	Set default mode (resolution) for playback.

The following command illustrates the use of these properties:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux! queue ! h264parse ! ! nvv4l2decoder ! nvdrmvideosink \
  conn_id=0 plane_id=1 set_mode=0 -e
```

nv3dsink Video Sink (Video playback using 3D graphics API)

This video sink element works with NVMM buffers and renders using the 3D graphics rendering API. It performs better than `nveglglessink` with NVMM buffers.

The following command starts the gstreamer pipeline using `nv3dsink`:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux ! queue ! h264parse ! nvv4l2decoder ! nv3dsink -e
```

The sink supports setting a specific window position and dimensions using the properties shown in this example:

```
nv3dsink --window-x=300 --window-y=300 --window-width=512 --window-
height=512"
```

Video Decode Support with OpenCV

You can simulate a video decode pipeline using the GStreamer-1.0 based OpenCV sample application `opencv_nvgstdec`.

Note: **The sample application currently operates only with OpenCV version 3.3.**

To perform video decoding with `opencv_nvgstdec`, enter the following command:

```
./opencv_nvgstdec --help
```

Note: **The `opencv_nvgstdec` application as distributed current supports only video decode of H264 format using the `nvv4l2decoder` plugin. You can modify and rebuild the application to support GStreamer pipelines for video decode of different formats. For display, the application utilizes an OpenCV based videosink component.**

For perform video decoding with `opencv_nvgstdec`, enter the command:

```
./opencv_nvgstdec --file-path=test_file_h264.mp4
```

VIDEO FORMAT CONVERSION WITH GSTREAMER-1.0

The NVIDIA proprietary `nvvidconv` GStreamer-1.0 plugin allows conversion between OSS (raw) video formats and NVIDIA video formats. The `nvvidconv` plugin currently supports the format conversions described in this section

raw-yuv Input Formats

Currently `nvvidconv` supports the I420, UYVY, YUY2, YVYU, NV12, GRAY8, BGRx, and RGBA raw-yuv input formats.

- Using the `gst-omx` encoder:

```
gst-launch-1.0 videotestsrc ! 'video/x-raw, format=(string)UYVY, \
  width=(int)1280, height=(int)720' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420’ ! omxh264enc ! \
  'video/x-h264, stream-format=(string)byte-stream' ! h264parse ! \
  qtmux ! filesink location=test.mp4 -e
```

- Using the `gst-v4l2` encoder (with other than the GRAY8 pipeline):

```
gst-launch-1.0 videotestsrc ! 'video/x-raw, format=(string)UYVY, \
  width=(int)1280, height=(int)720' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420’ ! \
  nvv4l2h264enc ! 'video/x-h264, \
  stream-format=(string)byte-stream' ! h264parse ! \
  qtmux ! filesink location=test.mp4 -e
```

- Using the `gst-v4l2` encoder with the GRAY8 pipeline:

```
gst -launch-1.0 videotestsrc ! 'video/x-raw, format=(string)GRAY8, \
  width=(int)640, height=(int)480, framerate=(fraction)30/1' ! \
  nvvidconv ! 'video/x-raw(memory:NVMM), format=(string)I420' ! \
  nvv4l2h264enc ! 'video/x-h264, \
  stream-format=(string)byte-stream' ! h264parse ! qtmux ! \
  filesink location=test.mp4 -e
```

raw-yuv Output Formats

Currently `nvvidconv` supports the I420, UYVY, YUY2, YVYU, NV12, GRAY8, BGRx, and RGBA raw-yuv output formats.

- Using the `gst-omx` decoder:

```
gst-launch-1.0 filesrc location=640x480_30p.mp4 ! qtdemux ! \
  queue ! h264parse ! omxh264dec ! nvvidconv ! \
  'video/x-raw, format=(string)UYVY' ! videoconvert ! xvimagesink -e
```

- Using the `gst-v4l2` decoder (with other than the GRAY8 pipeline):

```
gst-launch-1.0 filesrc location=640x480_30p.mp4 ! qtdemux ! \
  queue ! h264parse ! nvv4l2decoder ! nvvidconv ! \
  'video/x-raw, format=(string)UYVY' ! videoconvert ! xvimagesink -e
```

- Using the `gst-v4l2` decoder with the GRAY8 pipeline:

```
gst -launch-1.0 filesrc location=720x480_30i_MP.mp4 ! qtdemux ! \
queue ! h264parse ! nvv4l2decoder ! nvvidconv ! 'video/x-raw, \
format=(string)GRAY8' ! videoconvert ! xvimagesink -e
```

NVIDIA INPUT AND OUTPUT FORMATS

Currently `nvvidconv` supports NVIDIA input and output formats for format conversion as described in the following table:

Input Format	Output Format
NV12	NV12
I420 I420_10LE I420_12LE P010_10LE	I420 I420_10LE
UYVY YUY2 YVYU BGRx RGBA GRAY8	UYVY YUY2 YVYU BGRx RGBA GRAY8

Uses these commands to convert between NVIDIA formats:

- Using the `gst-omx` decoder:

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! \
  h264parse ! omxh264dec ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)RGBA' ! nvoverlaysink -e
```

- Using the `gst-v4l2` decoder:

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! \
  h264parse ! nvv4l2decoder ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)RGBA' ! nvdrmvideosink -
e
```

- Using the `gst-omx` encoder:

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420’ ! omxh264enc ! \
  qtmux ! filesink location=test.mp4 -e
```

- Using the `gst-v4l2` encoder:

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)I420’ ! nvv4l2h264enc !
\ h264parse ! qtmux ! filesink location=test.mp4 -e
```

- Using the `gst-v4l2` decoder and `nv3dsink` with the GRAY8 pipeline:

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! \
  h264parse ! nvv4l2decoder ! nvvidconv ! \
  'video/x-raw(memory:NVMM), format=(string)GRAY8' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), \ format=(string)I420' ! nv3dsink -e
```

VIDEO SCALING WITH GSTREAMER-1.0

The NVIDIA proprietary `nvvidconv` GStreamer-1.0 plugin also allows you to perform video scaling. The `nvvidconv` plugin currently supports scaling with the format conversions described in this section.

raw-yuv Input Formats

Currently `nvvidconv` supports the I420, UYVY, YUY2, YVYU, NV12, GRAY8, BGRx, and RGBA raw-yuv input formats for scaling.

- Using the `gst-omx` encoder:

```
gst-launch-1.0 videotestsrc ! \
  'video/x-raw, format=(string)I420, width=(int)1280, \
  height=(int)720' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)I420' ! omxh264enc ! \
  'video/x-h264, stream-format=(string)byte-stream' ! h264parse ! \
  qtmux ! filesink location=test.mp4 -e
```

- Using the `gst-v4l2` encoder:

```
gst-launch-1.0 videotestsrc ! \
  'video/x-raw, format=(string)I420, width=(int)1280, \
  height=(int)720' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)I420' ! nvv4l2h264enc ! \
  'video/x-h264, stream-format=(string)byte-stream' ! h264parse ! \
  qtmux ! filesink location=test.mp4 -e
```

raw-yuv Output Formats

Currently `nvvidconv` supports the I420, UYVY, YUY2, YVYU, NV12, BGRx, and RGBA raw-yuv output formats for scaling.

- Using the `gst-omx` decoder:

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! \
  queue ! h264parse ! omxh264dec ! nvvidconv ! \
  'video/x-raw, format=(string)I420, width=640, height=480' ! \
  xvimagesink -e
```

- Using the `gst-v4l2` decoder:

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! \
  queue ! h264parse ! nvv4l2decoder ! nvvidconv ! \
  'video/x-raw, format=(string)I420, width=640, height=480' ! \
  xvimagesink -e
```

NVIDIA Input and Output Formats

Currently `nvvidconv` supports the NVIDIA input and output formats for scaling described in the following table:

Input Format	Output Format
NV12	NV12
I420 I420_10LE I420_12LE P010_10LE	I420 I420_10LE
UYVY YUY2 YVYU BGRx	UYVY YUY2 YVYU BGRx

Input Format	Output Format
RGBA	RGBA
GRAY8	GRAY8

To scale between NVIDIA formats

- Using the `gst-omx` decoder:

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! \
  h264parse ! omxh264dec ! nvvidconv ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)NV12' ! nvoverlaysink -e
```

- Using the `gst-v4l2` decoder:

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! \
  h264parse ! nvv4l2decoder ! nvvidconv ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)NV12' ! nv3dsink -e
```

- Using the `gst-omx` decoder:

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! \
  h264parse ! omxh264dec ! nvvidconv ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)RGBA' ! nvoverlaysink -e
```

- Using the `gst-v4l2` decoder:

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! \
  h264parse ! nvv4l2decoder ! nvvidconv ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)NV12' ! nvdrmvideosink -e
```

- Using the `gst-omx` encoder:

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)NV12’ ! omxh264enc ! \
  qtmux ! filesink location=test.mp4 -e
```

- Using the `gst-v4l2` encoder:

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! nvvidconv ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)NV12’ ! nvv4l2h264enc ! h264parse ! \
  qtmux ! filesink location=test.mp4 -e
```

VIDEO CROPPING WITH GSTREAMER-1.0

The NVIDIA proprietary nvvidconv GStreamer-1.0 plugin also allows you to perform video cropping.

To crop video

- Using the `gst-omx` decoder:

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4> ! qtdemux ! \
  h264parse ! omxh264dec ! \
  nvvidconv left=400 right=1520 top=200 bottom=880 ! \
  nvoverlaysink display-id=1 -e
```

- Using the `gst-v4l2` decoder:

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4> ! qtdemux ! \
  h264parse ! nvv4l2decoder ! \
  nvvidconv left=400 right=1520 top=200 bottom=880 ! nv3dsink -e
```

VIDEO TRANSCODE WITH GSTREAMER-1.0

You can perform video transcoding between the following video formats.

H.264 Decode to VP9 Encode (NVIDIA Accelerated Decode to NVIDIA-Accelerated Encode)

- Using the `gst-omx` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h264parse ! \
  omxh264dec ! omxvp9enc bitrate=20000000 ! matroskamux name=mux ! \
  filesink location=<Transcoded_filename.mkv> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4> ! qtdemux ! \
  h264parse ! nvv4l2decoder ! \
  nvvidconv left=400 right=1520 top=200 bottom=880 ! nv3dsink -e
```

H.265 Decode to VP9 Encode (NVIDIA Accelerated Decode to NVIDIA-Accelerated Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h265parse ! \
  omxh265dec ! omxvp9enc bitrate=20000000 ! matroskamux name=mux ! \
  filesink location=<Transcoded_filename.mkv> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
 qtdemux name=demux demux.video_0 ! queue ! h265parse !
nvv4l2decoder \ ! nvv4l2vp9enc bitrate=20000000 ! queue !
matroskamux name=mux ! \
  filesink location=<Transcoded_filename.mkv> -e
```

VP8 Decode to H.264 Encode (NVIDIA Accelerated Decode to NVIDIA-Accelerated Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.webm> ! \
  matroskademux name=demux demux.video_0 ! queue ! omxvp8dec ! \
  omxh264enc bitrate=20000000 ! qtmux name=mux ! \
  filesink location=<Transcoded_filename.mp4> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mebm> ! \
  matroskademux name=demux demux.video_0 ! queue ! nvv4l2decoder ! \
  nvv4l2h264enc bitrate=20000000 ! h264parse ! queue ! \
  qtmux name=mux \ ! filesink location=<Transcoded_filename.mp4> -e
```

VP9 Decode to H.265 Encode (NVIDIA Accelerated Decode to NVIDIA-Accelerated Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.webm> ! \
  matroskademux name=demux demux.video_0 ! queue ! omxvp9dec ! \
  omxh265enc bitrate=20000000 ! qtmux name=mux ! \
  filesink location=<Transcoded_filename.mp4> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.webm> ! \
  matroskademux name=demux demux.video_0 ! queue ! nvv4l2decoder ! \
  nvv4l2h265enc bitrate=20000000 ! h265parse ! queue ! \
  qtmux name=mux \ ! filesink location=<Transcoded_filename.mp4> -e
```

MPEG-4 Decode to VP9 Encode (NVIDIA Accelerated Decode to NVIDIA-Accelerated Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! mpeg4videoparse ! \
  omxmpeg4videodec ! omxvp9enc bitrate=20000000 ! matroskamux \
  name=mux ! filesink location=<Transcoded_filename.mkv> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! mpeg4videoparse ! \
  nvv4l2decoder ! nvv4l2vp9enc bitrate=20000000 ! queue ! \
  matroskamux \   name=mux ! filesink \
  location=<Transcoded_filename.mkv> -e
```

MPEG-4 Decode to H.264 Encode (NVIDIA Accelerated Decode to NVIDIA-Accelerated Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! mpeg4videoparse ! \
  omxmpeg4videodec ! omxh264enc bitrate=20000000 ! \
  qtmux name=mux ! filesink location=<Transcoded_filename.mp4> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! mpeg4videoparse ! \
  nvv4l2decoder ! nvv4l2h264enc bitrate=20000000 ! h264parse ! \
  queue ! qtmux name=mux ! filesink \
  location=<Transcoded_filename.mp4> -e
```

H.264 Decode to VP8 Encode (NVIDIA Accelerated Decode to NVIDIA-Accelerated Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h264parse ! \
  omxh264dec ! omxvp8enc bitrate=20000000 ! queue ! \
  matroskamux name=mux ! \
  filesink location=<Transcoded_filename.mkv> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h264parse ! \
  nvv4l2decoder \ ! nvv4l2vp8enc bitrate=20000000 ! queue ! \
  matroskamux name=mux ! \
  filesink location=<Transcoded_filename.mkv> -e
```

H.265 Decode to VP8 Encode (NVIDIA Accelerated Decode to NVIDIA-Accelerated Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h265parse ! \
  omxh265dec ! \ omxvp8enc bitrate=20000000 ! queue ! \
  matroskamux name=mux ! \
  filesink location=<Transcoded_filename.mkv> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h265parse ! \
  nvv4l2decoder \ ! nvv4l2vp8enc bitrate=20000000 ! queue ! \
  matroskamux name=mux ! \
  filesink location=<Transcoded_filename.mkv> -e
```

VP8 Decode to MPEG-4 Encode (NVIDIA Accelerated Decode to OSS Software Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.mkv> ! \
  matroskademux name=demux demux.video_0 ! queue ! omxvp8dec ! \
  nvvidconv ! avenc_mpeg4 bitrate=4000000 ! queue ! \
  qtmux name=mux ! filesink location=<Transcoded_filename.mp4> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mkv> ! \
  matroskademux name=demux demux.video_0 ! queue ! nvv4l2decoder ! \
  nvvidconv ! avenc_mpeg4 bitrate=4000000 ! queue ! \
  qtmux name=mux ! filesink location=<Transcoded_filename.mp4> -e
```

VP9 Decode to MPEG-4 Encode (NVIDIA Accelerated Decode to OSS Software Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.mkv> ! \
  matroskademux name=demux demux.video_0 ! queue ! omxvp9dec ! \
  nvvidconv ! avenc_mpeg4 bitrate=4000000 ! qtmux name=mux ! \
  filesink location=<Transcoded_filename.mp4> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mkv> ! \
  matroskademux name=demux demux.video_0 ! queue ! nvv4l2decoder ! \
  nvvidconv ! avenc_mpeg4 bitrate=4000000 ! qtmux name=mux ! \
  filesink location=<Transcoded_filename.mp4> -e
```

H.264 Decode to Theora Encode (NVIDIA Accelerated Decode to OSS Software Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h264parse ! \
  omxh264dec ! nvvidconv ! theoraenc bitrate=4000000 ! \
  oggmux name=mux ! filesink location=<Transcoded_filename.ogg> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
 qtdemux name=demux demux.video_0 ! queue ! h264parse ! \
  nvv4l2decoder \ ! nvvidconv ! theoraenc bitrate=4000000 ! \
  oggmux name=mux ! filesink location=<Transcoded_filename.ogg> -e
```

H.264 Decode to H.263 Encode (NVIDIA Accelerated Decode to OSS Software Encode)

- Using the `gst-omx`-pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h264parse ! \
  omxh264dec ! nvvidconv ! \
  'video/x-raw, width=(int)704, height=(int)576, \
  format=(string)I420' ! avenc_h263 bitrate=4000000 ! qtmux ! \
  filesink location=<Transcoded_filename.mp4> -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h264parse ! \
  nvv4l2decoder \  ! nvvidconv ! \
  'video/x-raw, width=(int)704, height=(int)576, \
  format=(string)I420' ! avenc_h263 bitrate=4000000 ! qtmux ! \
  filesink location=<Transcoded_filename.mp4> -e
```

CUDA VIDEO POST-PROCESSING WITH GSTREAMER-1.0

This section describes GStreamer-1.0 plugins for NVIDIA® CUDA® post-processing operations.

gst-videocuda

This GStreamer-1.0 plugin performs CUDA post-processing operations on decoder-provided EGL images and render video using `nveglglessink`.

The following are sample pipeline creation and application usage commands.

Sample decode pipeline

```
gst-launch-1.0 filesrc location=<filename_h264_1080p.mp4> ! \
  qtdemux name=demux ! h264parse ! omxh264dec ! videocuda !\
  nveglglessink max-lateness=-1 -e
```

Sample decode command

```
nvgstplayer-1.0 -i <filename_h264_1080p.mp4> --svd="omxh264dec" \
  --svc="videocuda" --svs="nveglglessink # max-lateness=-1" \
  --disable-vnative --no-audio --window-x=0 --window-y=0 \
  --window-width=960 --window-height=540
```

gst-nvivafilter

This NVIDIA proprietary GStreamer-1.0 plugin performs pre/post and CUDA post-processing operations on CSI camera captured or decoded frames, and renders video using overlay video sink or video encode.

Note **The `gst-nvivafilter` pipeline requires unsetting the `DISPLAY` environment variable using the command `unset DISPLAY` if lightdm is stopped.**

Sample decode pipeline

- Using the `gst-omx`-decoder:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! qtdemux ! \
  h264parse !  omxh264dec ! nvivafilter cuda-process=true \
  customer-lib-name="libnvsample_cudaprocess.so" ! \
  'video/x-raw(memory:NVMM), format=(string)NV12' ! nvoverlaysink -e
```

- Using the `gst-v4l2` decoder:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! qtdemux ! queue ! \
  h264parse !  nvv4l2decoder ! nvivafilter cuda-process=true \
  customer-lib-name="libnvsample_cudaprocess.so" ! \
  'video/x-raw(memory:NVMM), format=(string)NV12' ! nvdrmvideosink \
  -e
```

Sample CSI Camera pipeline

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)3840, height=(int)2160, \
  format=(string)NV12, framerate=(fraction)30/1' ! \
  nvivafilter cuda-process=true \
  customer-lib-name="libnvsample_cudaprocess.so" ! \
  'video/x-raw(memory:NVMM), format=(string)NV12' ! nvoverlaysink -e
```

Note **See the `nvsample_cudaprocess_src.tbz2` package for the `libnvsample_cudaprocess.so` library sources. A sample CUDA implementation of `libnvsample_cudaprocess.so` can be replaced by a custom CUDA implementation.**

VIDEO ROTATION WITH GSTREAMER-1.0

The NVIDIA proprietary `nvvidconv` GStreamer-1.0 plugin also allows you to perform video rotation operations.

The following table shows the supported values for the `nvvidconv` `flip-method` property.

Flip Method	Property value
Identity - no rotation (default)	0
Counterclockwise - 90 degrees	1
Rotate - 180 degrees	2
Clockwise - 90 degrees	3
Horizontal flip	4

Flip Method	Property value
Upper right diagonal flip	5
Vertical flip	6
Upper-left diagonal	7

Note **Get information on the `nvvidconv` flip-method property with the `gst-inspect-1.0 nvvidconv` command.**

To rotate video 90 degrees counterclockwise

To rotate video 90 degrees in a counterclockwise direction, enter the following command.

- With the `gst-omx`-decoder:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux ! h264parse ! omxh264dec ! \
  nvvidconv flip-method=1 ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvoverlaysink -e
```

- With the `gst-v4l2` decoder:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux ! h264parse ! nvv4l2decoder ! \
  nvvidconv flip-method=1 ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! \
  nvdrmvideosink -e
```

To rotate video 90 degrees clockwise

To rotate video 90 degrees in a clockwise direction, enter the following command:

```
gst-launch-1.0 filesrc location=<filename.mp4> ! qtdemux name=demux ! \
  h264parse ! omxh264dec ! nvvidconv flip-method=3 ! \
  'video/x-raw(memory:NVMM), format=(string)I420’ ! \
  omxh264enc ! qtmux ! filesink location=test.mp4 -e
```

To rotate 180 degrees

To rotate video 180 degrees, enter the following command:

```
gst-launch-1.0 nvarguscamerasrc! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! \
  nvvidconv flip-method=2 ! \
  'video/x-raw(memory:NVMM), format=(string)I420' ! nvoverlaysink -e
```

To scale and rotate video 90 degrees counterclockwise

To scale and rotate video 90 degrees counterclockwise, enter the following command:

- Using the `gst-omx`-decoder:

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4> ! qtdemux ! \
  h264parse ! omxh264dec ! nvvidconv flip-method=1 ! \
  'video/x-raw(memory:NVMM), width=(int)480, height=(int)640, \
  format=(string)I420' ! nvoverlaysink -e
```

- Using the `gst-v4l2` decoder:

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4> ! qtdemux ! \
  h264parse ! nvv4l2decoder ! nvvidconv flip-method=1 ! \
  'video/x-raw(memory:NVMM), width=(int)480, height=(int)640, \
  format=(string)I420' ! nvdrmvideosink -e
```

To scale and rotate video 90 degrees clockwise

To scale and rotate video 90 degrees clockwise, enter the following command:

```
gst-launch-1.0 nvarguscamerasrc ! \
  'video/x-raw(memory:NVMM), width=(int)1920, height=(int)1080, \
  format=(string)NV12, framerate=(fraction)30/1' ! \
  nvvidconv flip-method=3 ! 'video/x-raw(memory:NVMM), \
  width=(int)480, height=(int)640, format=(string)I420' ! \
  nvoverlaysink -e
```

To scale and rotate video 180 degrees

To scale and rotate video 180 degrees, enter the following command:

- Using the `gst-omx` decoder:

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4> ! \
  qtdemux ! h264parse ! omxh264dec ! nvvidconv flip-method=2 ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)I420' ! nvoverlaysink -e
```

- Using the `gst-v4l2` decoder:

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4> ! \
  qtdemux ! h264parse ! nvv4l2decoder ! nvvidconv flip-method=2 ! \
  'video/x-raw(memory:NVMM), width=(int)640, height=(int)480, \
  format=(string)I420' ! nvdrmvideosink -e
```

VIDEO COMPOSITION WITH GSTREAMER-1.0

With the NVIDIA proprietary nvcompositor GStreamer-1.0 plugin, you can perform video composition operations on gst-omx video decoded streams.

Note **`nvcompositor` supports video decode (gst-omx) with the overlay render pipeline for gst-1.14.**

Prerequisites

- Install the following dependent GStreamer package.

```
$ sudo apt-get install gstreamer1.0-plugins-bad
```

- Clear the registry cache file, in case there is an issue with `gst-inspect-1.0 nvcompositor`.

```
$ rm .cache/gstreamer-1.0/registry.aarch64.bin
```

To composite decoded streams with different formats

Enter the following command:

- Using the `gst-omx` decoder:

```
gst-launch-1.0 nvcompositor \
  name=comp sink_0::xpos=0 sink_0::ypos=0 sink_0::width=1920 \
  sink_0::height=1080 sink_1::xpos=0 sink_1::ypos=0 \
  sink_1::width=1600 sink_1::height=1024 sink_2::xpos=0 \
  sink_2::ypos=0 sink_2::width=1366 sink_2::height=768 \
  sink_3::xpos=0 sink_3::ypos=0 sink_3::width=1024 \
  sink_3::height=576 ! nvoverlaysink display-id=1 \
  filesrc location=<filename_h264_1080p_30fps.mp4> ! qtdemux ! \
  h264parse ! omxh264dec ! comp. filesrc \
  location=< filename_h265_1080p_30fps.mp4> ! qtdemux ! h265parse !
\
  omxh265dec ! comp. filesrc \
  location=< filename_vp8_1080p_30fps.webm>  matroskademux ! \
  omxvp8dec ! \
  comp. filesrc location=<filename_vp9_1080p_30fps.webm> ! \
  matroskademux ! omxvp9dec ! comp. -e
```

- Using the `gst-v4l2` decoder:

```
gst-launch-1.0 nvcompositor \
  name=comp sink_0::xpos=0 sink_0::ypos=0 sink_0::width=1920 \
  sink_0::height=1080 sink_1::xpos=0 sink_1::ypos=0 \
  sink_1::width=1600 sink_1::height=1024 sink_2::xpos=0 \
  sink_2::ypos=0 sink_2::width=1366 sink_2::height=768 \
  sink_3::xpos=0 sink_3::ypos=0 sink_3::width=1024 \
  sink_3::height=576 ! nv3dsink \
  filesrc location=<filename_h264_1080p_30fps.mp4> ! qtdemux ! \
  h264parse ! nvv4l2decoder ! comp. filesrc \
  location=< filename_h265_1080p_30fps.mp4> ! qtdemux ! \
  h265parse ! \ nvv4l2decoder ! comp. filesrc \
  location=< filename_vp8_1080p_30fps.webm> ! matroskademux ! \
  nvv4l2decoder ! \
  comp. filesrc location=<filename_vp9_1080p_30fps.webm> !\
  matroskademux ! nvv4l2decoder ! comp. -e
```

INTERPOLATION METHODS FOR VIDEO SCALING

The NVIDIA proprietary `nvvidconv` GStreamer-1.0 plugin allows you to choose the interpolation method used for scaling.

The following table shows the supported values for the `nvvidconv interpolation-method` property.

Interpolation Method	Property Value
Nearest	0
Bilinear	1
5-tap	2
10-tap	3
Smart (default)	4
Nicest	5

Note **Get information on the nvvidconv `interpolation-method` property with the `gst-inspect-1.0 nvvidconv` command.**

To use bilinear interpolation method for scaling

Enter the following command:

- Using the `gst-omx` pipeline:

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4>! \
  qtdemux name=demux ! h264parse ! omxh264dec ! \
  nvvidconv interpolation-method=1 ! \
  'video/x-raw(memory:NVMM), format=(string)I420, width=1280, \
  height=720' ! nvoverlaysink -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename_1080p.mp4>! \
  qtdemux name=demux ! h264parse ! nvv4l2decoder ! \
  nvvidconv interpolation-method=1 ! \
  'video/x-raw(memory:NVMM), format=(string)I420, width=1280, \
  height=720' ! nvdrmvideosink -e
```

EGLSTREAM PRODUCER EXAMPLE

The NVIDIA-proprietary nveglstreamsrc and nvvideosink GStreamer-1.0 plugins allow simulation of an EGLStream producer pipeline (for preview only.)

To simulate an EGLStream producer pipeline

Enter the following command:

```
nvgstcapture-1.0 --camsrc=3
```

EGL IMAGE TRANSFORM EXAMPLE

The NVIDIA proprietary nvegltransform GStreamer-1.0 plugin allows simulation of an EGLImage transform pipeline.

To simulate an EGL Image transform pipeline

Enter the following command:

- Using the `gst-omx` pipeline:

```
gst-launch-1.0 filesrc location=<filename_h264_1080p.mp4>  ! \
  qtdemux ! h264parse ! omxh264dec ! nvvidconv ! \
  'video/x-raw(memory:NVMM), width=(int)1280, height=(int)720, \
  format=(string)NV12' ! nvegltransform ! nveglglessink -e
```

- Using the `gst-v4l2` pipeline:

```
gst-launch-1.0 filesrc location=<filename_h264_1080p.mp4>  ! \
  qtdemux ! h264parse ! nvv4l2decoder ! nvegltransform !
nveglglessink -e
```

GSTREAMER BUILD INSTRUCTIONS

This release contains the `gst-install` script to install a specific GStreamer version. This section provides a procedure for building current versions of GStreamer.

To build GStreamer using gst-install

1. Execute the following command:

   ```
   gst-install [--prefix=<install_path>] [--version=<version>]
   ```

   Where `<install_path>` is the location where you are installing GStreamer and `<version>` is the GStreamer version. For example:

   ```
   gst-install --prefix=/home/ubuntu/gst-1.16.0 --version=1.16.0
   ```

2. Export environment variables with the following command:

   ```
   export LD_LIBRARY_PATH=<install_path>/lib/aarch64-linux-gnu
   export PATH=<install_path>/bin:$PATH
   ```

   Where `<install_path>` is the location where you are installing GStreamer. For example:

   ```
   export LD_LIBRARY_PATH=/home/ubuntu/gst-1.16.0/lib/aarch64-linux-gnu
   export PATH=/home/ubuntu/gst-1.16.0/bin:$PATH
   ```

To build GStreamer manually

1. Download the latest version of gstreamer available at:

   ```
   http://gstreamer.freedesktop.org/src/
   ```

   The following are the files you need from version 1.16.0:

   - gstreamer-1.16.0.tar.xz
   - gst-plugins-base-1.16.0.tar.xz
   - gst-plugins-good-1.16.0.tar.xz
   - gst-plugins-bad-1.16.0.tar.xz
   - gst-plugins-ugly-1.16.0.tar.xz

2. Install needed packages with the following command:

   ```
   sudo apt-get install build-essential dpkg-dev flex bison \
     autotools-dev automake liborc-dev autopoint libtool \
     gtk-doc-tools libgstreamer1.0-dev
   ```

3. In the `~/` directory, create a `gst_<version>` directory, where <version> is the version number of gstreamer you are building.
4. Copy the downloaded tar.xz files to the gst_<version> directory.
5. Uncompress the tar.xz files in the `gst_<version>` directory.
6. Set the `PKG_CONFIG_PATH` environment variable with the following command:

   ```
   export PKG_CONFIG_PATH=/home/ubuntu/gst_1.16.0/out/lib/pkgconfig
   ```

7. Build gstreamer (in this example, gstreamer-1.16.0) with the following commands:

   ```
   ./configure --prefix=/home/ubuntu/gst_1.16.0/out
   make
   make install
   ```

8. Build `gst-plugins-base-1.16.0` with the following commands:

   ```
   sudo apt-get install libxv-dev libasound2-dev libtheora-dev \
     libogg-dev libvorbis-dev
   ./configure --prefix=/home/ubuntu/gst_1.16.0/out
   make
   make install
   ```

9. Build gst-plugins-good-1.16.0 with the following commands:

```
sudo apt-get install libbz2-dev libv4l-dev libvpx-dev \
  libjack-jackd2-dev libsoup2.4-dev libpulse-dev
./configure --prefix=/home/ubuntu/gst_1.16.0/out
make
make install
```

10. Obtain and build gst-plugins-bad-1.16.0 with the following commands:

```
sudo apt-get install faad libfaad-dev libfaac-dev
./configure --prefix=/home/ubuntu/gst_1.16.0/out
make
make install
```

11. Obtain and build gst-plugins-ugly-1.16.0 with the following commands:

```
sudo apt-get install libx264-dev libmad0-dev
./configure --prefix=/home/ubuntu/gst_1.16.0/out
make
make install
```

12. Set the LD_LIBRARY_PATH environment variable with the following command:

```
export LD_LIBRARY_PATH=/home/ubuntu/gst_1.16.0/out/lib/
```

13. Copy the `nvidia gstreamer-1.0` libraries to the `gst_`1.16.0 plugin directory using the following command:

```
cd /usr/lib/aarch64-linux-gnu/gstreamer-1.0/
cp libgstnv* libgstomx.so \
  ~/gst_1.16.0/out/lib/gstreamer-1.0/
```

The `nvidia gstreamer-1.0` libraries include:

```
libgstnvarguscamera.so
libgstnvcompositor.so
libgstnvdrmvideosink.so
libgstnveglglessink.so
libgstnveglstreamsrc.so
libgstnvegltransform.so
libgstnvivafilter.so
libgstnvjpeg.so
libgstnvtee.so
libgstnvvidconv.so
```

```
libgstnvvideo4linux2.so
libgstnvvideocuda.so
libgstnvvideosink.so
libgstnvvideosinks.so
libgstomx.so
```

NVGSTCAPTURE-1.0 REFERENCE

This section describes the `nvgstcapture-1.0` application.

Note: **By default, the `nvgstcapture-1.0` application only supports the ARGUS API using the `nvarguscamerasrc` plugin. The legacy `nvcamerasrc` plugin is no longer supported.**

COMMAND LINE OPTIONS

The nvgstcapture-1.0 application can display information about its own command line options. To display command line option information, run the application with one of these command line options:

“Help” Command Line Options	
Option	**Description**
-h --help	Show help options except for GStreamer options.
--help-all	Show all help options.
--help-gst	Show GStreamer options.

This table describes the application’s other command-line options.

Other Command Line Options		
Option	**Description**	**Notes** ***Examples***
--prev_res	Preview width and height.	Range: 2 to 8 (3840x2160) *--prev_res=3*
--cus-prev-res	Custom preview width and height for CSI only.	*--cus-prev-res=1920x1080*

Other Command Line Options		
Option	**Description**	**Notes** ***Examples***
--image_res	Image width and height,	Range: 2 to 12 (5632x4224) *--image_res=3*
--video-res	Video width and height.	Range: 2 to 9 (3896x2192) *--video-res=3*
--camsrc	Camera source to use	0=V4L2 1=csi[default] 2=videotest 3=eglstream
-m, --mode	Capture mode.	1-Still 2-Video
-v, --video_enc	Video encoder type.	0=h264[HW] 1=vp8[HW, not supported on Jetson AGX Xavier] 2=h265[HW] 3=vp9[HW]
-p, --hw-enc-path	Framework Type.	0=OMX 1=V4L2
-b, --enc-bitrate	Video encoding Bit-rate (in bytes)	*--enc-bitrate=4000000*
--enc-controlrate	Video encoding bit-rate control method.	0 = Disable 1 = variable (Default) 2 = constant *--enc-controlrate=1*
--enc-EnableTwopassCBR	Enable two pass CBR while encoding.	0 = Disable 1 = Enable *--enc-EnableTwopassCBR=1*
--enc-profile	Video encoder profile (only for H.264).	0-Baseline 1-Main 2-High
-j, --image_enc	Image encoder type.	0-jpeg_SW[jpegenc] 1-jpeg_HW[nvjpegenc]
-k, --file_type	Container file type.	0-MP4 1-3GP 2-MKV
--file-name	Captured file name. nvcamtest is used by default.	
--color-format	Color format to use.	0=I420 1=NV12[For CSI only and default for CSI] 2=YUY2[For V4L2 only, default for V4L2]
--orientation	Camera sensor orientation value.	

Other Command Line Options		
Option	**Description**	**Notes** ***Examples***
--eglConfig	EGL window Coordinates (x_pos y_pos) in that order.	*--eglConfig="50 100"*
-w, --whitebalance	Capture whitebalance value.	
--timeout	Capture timeout value.	
--saturation	Camera saturation value.	
--sensor-id	Camera Sensor ID value.	
--display-id	[For nvoverlaysink only] Display ID value.	
--overlayConfig	Overlay Configuration Options index and coordinates in (index, x_pos, y_pos, width, height) order.	*--overlayConfig="0, 0, 0, 1280, 720"*
--cap-dev-node	Video capture device node.	0=/dev/video0[default] 1=/dev/video1 2=/dev/video2 *--cap-dev-node=0*
--svs=<chain>	Where `<chain>` is a chain of GStreamer elements: For USB, specifies a chain for video preview. For CSI only, use `nvoverlaysink` or `nvdrmvideosink`.	
--exposuretimerange	Property to adjust exposure time range in nanoseconds.	*--exposuretimerange="34000 358733000"*
--gainrange	Property to adjust gain range	*--gainrange="1 16*
--ispdigitalgainrange	Property to adjust digital gain range.	*--ispdigitalgainrange="1 8"* *Range value from 1 to 256*
--aelock	Enable AE Lock.	Default is disabled.
--awblock	Enable AWB Lock.	Default is disabled.
--exposurecompensation	Property to adjust exposure compensation.	Range value from -2.0 to 2.0. *--exposurecompensation=0.5*
--aeantibanding	Property to set the auto exposure antibanding mode.	Range value from 0 to 3. *--aeantibanding=2*
--tnr-mode	Property to select temporal noise reduction mode.	*--tnr-mode=2*
--tnr-strength	Property to adjust temporal noise reduction strength.	*--tnr-strength=0.5*
--ee-mode	Property to select edge enhancement mode.	*--ee-mode=2*

Other Command Line Options		
Option	Description	Notes ***Examples***
--ee-strength	Property to adjust edge enhancement strength.	*--ee-strength=0.5*

CSI CAMERA SUPPORTED RESOLUTIONS

CSI camera supports the following image resolutions for Nvarguscamera:

- 640x480
- 1280x720
- 1920x1080
- 2104x1560
- 2592x1944
- 2616x1472
- 3840x2160
- 3896x2192
- 4208x3120
- 5632x3168
- 5632x4224

CSI CAMERA RUNTIME COMMANDS

Options for Nvarguscamera

CSI camera runtime commands options for Nvarguscamera are described in the following table.

Command	Description	Notes
h	Help	—
q	Quit	—
mo:<value>	Set capture mode	1-image 2-video
gmo	Get capture mode	—
so:<val>	Set sensor orientation	0-none 1-Rotate counter-clockwise 90 degrees 2-Rotate 180 degrees 3-Rotate clockwise 90 degrees
gso	Get sensor orientation	—

Command	Description	Notes
wb:<value>	Set white balance mode	0-off 1-auto 2-incandescent 3-fluorescent 4-warm-fluorescent 5-daylight 6-cloudy-daylight 7-twilight 8-shade 9-manual
gwb	Get white balance mode	—
st:<value>	Set saturation	0-2, e.g., st:1.25
gst	Get saturation	—
j	Capture one image.	—
jx<delay>	Capture after a delay of <delay>, e.g., jx5000 to capture after a 5-second delay	—
j:<value>	Capture <count> number of images in succession, e.g., j:6 to capture 6 images.	—
0	Stop recording video	—
1	Start recording video	—
2	Video snapshot (while recording video)	—
gpcr	Get preview resolution	—
gicr	Get image capture resolution	—
gvcr	Get video capture resolution	—

USB CAMERA RUNTIME COMMANDS

USB Camera Runtime Commands

USB camera runtime commands are described in the following table.

Command	Description	Notes
h	Help	—
q	Quit	—
mo:<value>	Set capture mode	1-image 2-video
gmo	Get capture mode	—
j	Capture one image.	—

Command	Description	Notes
jx<delay>	Capture after a delay of <delay>, e.g., jx5000 to capture after a 5-second delay	—
j:<value>	Capture <count> number of images in succession, e.g., j:6 to capture 6 images.	—
1	Start recording video	—
0	Stop recording video	—
pcr:<value>	Set preview resolution	0-176x144 1-320x240 2-640x480 3-1280x720
gpcr	Get preview resolution	—
gicr	Get image capture resolution	—
gvcr	Get video capture resolution	—
br:<value>	Set encoding bit rate (in bytes)	e.g., br:4000000
gbr	Get encoding bit rate	—
cdn:<value>	Set capture device node	0-/dev/video0 1-/dev/video1 2-/dev/video2
gcdn	Get capture device node	—

Runtime Video Encoder Configuration Options

The following table describes runtime video encoder configuration options supported for Nvarguscamera.

Command	Description	Notes
br:<val>	Sets encoding bit-rate (in bytes)	Example: br:4000000
gbr	Gets encoding bit-rate (in bytes)	—
ep:<val>	Sets encoding profile (for H.264 only)	Example: ep:1 (0): Baseline (1): Main (2): High
gep	Gets encoding profile (for H.264 only)	—
Enter 'f'	Forces IDR frame on video encoder (for H.264 only)	—

NOTES

- The nvgstcapture-1.0 application generates image and video output files in the same directory as the application itself.
- Filenames for image and video content are in the formats, respectively:
  - `nvcamtest_<pid>_<sensor_id>_<counter>.jpg`
  - `nvcamtest_<pid>_<sensor_id>_<counter>.mp4`

  Where:

  - <pid> is the process ID.
  - <sensor_id> is the sensor ID.
  - <counter> is a counter starting from 0 every time you run the application.

  Rename or move files between runs to avoid overwriting results you want to save.
- The nvgstcapture-1.0 application supports native capture(video only) mode by default.
- Advance features, like setting zoom, brightness, exposure, and whitebalance levels, are not supported for USB camera.

NVGSTPLAYER-1.0 REFERENCE

This section describes the operation of the the `nvgstplayer-1.0` application.

NVGSTPLAYER COMMAND LINE OPTIONS

Note: **To list supported options, enter the command:**

```
nvgstplayer-1.0 --help
```

This table describes `nvgstplayer-1.0` command-line options:

Option	Description and Example
-u <path> --urifile <path>	Path of the file containing the URIs. Example: `-u my_uri.txt`

-I <uri>
--uri <uri>
	Input URI.
	Examples:
	`-uri file:///home/ubuntu/movie.avi`
	`-uri http://www.joedoe.com/foo.ogg`

-e <path>
--elemfile <path>
	Element(s) (Properties) file.
	The element file may contain an audio or video processing elements chain like this:
	`[sas]`
	`pipe=alsasink # device=demixer`

-x --cxpr	Command sequence expression. Example: `-cxpr="r5 s0"`
-n <n> --loop <n>	Number of times to play the media.
-c <n> --audio-track <n>	If a stream has multiple audio tracks, specifies the track number to play.

Option	Description and Example
-v <n> --video-track <n>	If a stream has multiple video tracks, specifies the track number to play.
-a <seconds> --start <seconds>	Point to start playback, in seconds from the beginning of the media segment.
-d <seconds> --duration <seconds>	Duration of playback, in seconds.
--no-sync	Disable AV sync.
--disable-dpms	Unconditionally disable DPMS/ScreenBlanking during operation; re-enable on exit.
--stealth	Operate in stealth mode, staying alive even when no media is playing.
--bg	Operate in background mode, ignoring keyboard input.
--use-playbin	Use Playbin GStreamer element.
--no-audio	Disable audio.
--no-video	Disable video.
--disable-anative	Disable native audio rendering.
--disable-vnative	Disable native video rendering.
--use-buffering	Enable `decodebin` property for emit of `GST_MESSAGE_BUFFERING` based on low and high percent thresholds.
-l <percent> --low-percent <percent>	Low threshold for buffering to start, in percent.
-j <percent> --high-percent <percent>	High threshold for buffering to finish, in percent.
--loop-forever	Play the URI(s) in an endless loop.
-t <seconds> --max-size-time <seconds>	Maximum time in queue, in seconds (0=automatic).
-y <n> --max-size-bytes <n>	Maximum amount of memory in the queue, in bytes (0=automatic).
-b <n> --max-size-buffers <n>	Maximum number of buffers in the queue (0=automatic).
--window-x <n>	X coordinate for player window (for non-overlay rendering).
--window-y <n>	Y coordinate for player window (for non-overlay rendering).
--window-width <n>	Window width (for non-overlay rendering).
--window-height <n>	Window height (for non-overlay rendering).
--disable-fullscreen	Play video in non-full-screen mode (for `nveglglessink`).

Option		Description and Example
-k <seconds> --image-display-time <seconds>		Image display time, in seconds.
--show-tags		Shows tags (metadata), if available.
--stats		Shows stream statistics, if enabled.
--stats-file		File to dump stream statistics, if enabled.
--svd=<chain>	Where `<chain>` is a chain of GStreamer elements	Chain to use for video decoding.
--sad=<chain>		Chain to use for audio decoding.
--svc=<chain>		Chain to use for video postprocessing.
--sac=<chain>		Chain to use for audio postprocessing.
--svs=<chain>		Chain to use for video rendering.
--sas=<chain>		Chain to use for audio rendering.
--shttp=<chain>		Chain to use for http source.
--srtsp=<chain>		Chain to use for rtsp source.
--sudp=<chain>		Chain to use for udp source.
--sfsrc=<chain>		Chain to use for file source.

NVGSTPLAYER RUNTIME COMMANDS

This table describes nvgstplayer runtime commands.

Command	Description ***Example***
h	Help
q	Quit
Up Arrow]	Go to next track.
c	Restart current track.
Down Arrow [	Go to previous track.
spos	Query for position (time from start).
sdur	Query for duration.
s<n>	Seek to `<n>` seconds from start. *e.g.* `s5.120`
v<pct>	Seek to `<pct>` percent of the duration. e.g. v54

Command	**Description** ***Example***
f<val>	Shift <n> seconds relative to current position. e.g. `f23.901`
Left Arrow <	Seek backward 10 seconds.
Right Arrow >	Seek forward 10 seconds.
p	Pause playback.
r	Start/resume playback.
z	Stop playback.
i:<uri>	Enter a single URI.

VIDEO ENCODER FEATURES

The GStreamer-1.0-based `gst-omx` video encoders support the following features, respectively:

Video Encoder Feature	H264enc	H265enc	Vp8enc	Vp9enc
profile (Baseline / Main / High)	✓ (all)	✓ (Main)	✓	✓
level	✓	✓	—	—
bitrate	✓	✓	✓	✓
peak bitrate	✓	✓	—	—
stringent bitrate	✓	✓	—	—
insert-spsppsatidr	✓	✓	✓	✓
control-rate	✓	✓	✓	✓
iframeinterval	✓	✓	✓	✓
qp-range	✓	✓	✓	✓
temporal-tradeoff	✓	✓	✓	✓
bit-packetization	✓	✓	✓	✓
preset-level	✓	✓	✓	✓
low-latency	✓	✓	✓	✓
slice-header spacing	✓	✓	—	—
force-IDR	✓	✓	✓	✓
vbv-size	✓	✓	✓	✓
sliceintrarefreshenable	✓	✓	—	—
sliceintrarefreshinterval	✓	✓	—	—
EnableTwoPassCBR	✓	✓	✓	✓
num-B-Frames	✓	—	—	—

The GStreamer-1.0-based gst-v4l2 video encoders support the following features, respectively :

Video Encoder Feature	H264enc	H265enc	Vp8enc	Vp9enc
profile (Baseline / Main / High)	✓ (all)	✓ (Main)	✓	✓
control-rate	✓	✓	✓	✓
bitrate	✓	✓	✓	✓
insert-spsppsatidr	✓	—	—	—
profile	✓	—	—	—
quantization range for I, P and B frame	✓	✓	—	—
iframeinterval	✓	✓	✓	✓
qp-range	✓	✓	—	—
bit-packetization	✓	✓	—	—
preset-level	✓	✓	✓	✓
slice-header spacing	✓	✓	—	—
force-IDR	✓	✓	✓	✓
EnableTwoPassCBR	✓	✓	—	—
Enable cabac-entropy-coding	✓	—	—	—
Enable MVBufferMeta	✓	✓	—	—
Insert aud	✓	✓	—	—
Insert vui	✓	✓	—	—
num-B-Frames	✓	—	—	—

SUPPORTED CAMERAS

This section describes the supported cameras.

CSI CAMERAS

- NVIDIA® Jetson Nano™, NVIDIA® Jetson AGX Xavier™, and NVIDIA® Jetson™ TX2 can capture camera images via CSI interface.
- Jetson Nano, Jetson AGX Xavier, and Jetson TX2 all support both YUV and RAW Bayer capture data.
- GStreamer supports simultaneous capture from multiple CSI cameras. Support is validated using the `nvgstcapture` application.
- Capture is validated for SDR, PWL HDR and DOL HDR modes for various sensors using the `nvgstcapture` application.
- Jetson AGX Xavier and Jetson TX2 also support the MIPI CSI virtual channel feature. The virtual channel is a unique channel identifier used for multiplexed sensor streams sharing the same CSI port/brick and CSI stream through supported GMSL (Gigabit Multimedia Serial Link) aggregators.
- GMSL + VC capture is validated on Jetson AGX Xavier and Jetson TX2 using the `nvgstcapture` application. The reference GMSL module (MAX9295-serializer/ MAX9296-deserializer/IMX390-sensor) is used for validation purposes.

USB 2.0 CAMERAS

The following camera has been validated on Tegra platforms running L4T with USB 2.0 ports. This camera is UVC compliant.

- Logitech C920

```
http://www.logitech.com/en-in/product/hd-pro-webcam-c920
```

INDUSTRIAL CAMERA DETAILS

The following USB 3.0 industrial camera is validated on Jetson AGX Xavier under L4T:

- See3CAM_CU130

  http://www.e-consystems.com/UltraHD-USB-Camera.asp

  - USB 3.0
  - UVC compliant
  - 3840 x 2160 at 30 FPS | 4224 x 3156 at 13 FPS
  - Purpose - Embedded Navigation
  - Test using the `nvgstcapture` app.
  - Issues encountered:
    - FPS cannot be fixed. Changes based on exposure.
    - FPS cannot be changed. Needs payment to vendor to get the support added to their firmware.