## DOCUMENT CHANGE HISTORY

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Authors</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1.0</td>
<td>01 May 2015</td>
<td>NVIDIA</td>
<td>Initial release.</td>
</tr>
<tr>
<td>v1.1</td>
<td>30 Jun 2015</td>
<td>mzensius</td>
<td>Added rotation and scaling commands, other new content.</td>
</tr>
<tr>
<td>v1.2</td>
<td>03 Nov 2015</td>
<td>emilyh</td>
<td>Changes for 23.1</td>
</tr>
<tr>
<td>v1.3</td>
<td>19 Nov 2015</td>
<td>mzensius</td>
<td>Added note for display export.</td>
</tr>
<tr>
<td>v1.4</td>
<td>17 Dec 2015</td>
<td>hlang</td>
<td>Updated gst-nvivafilter sample pipelines. Updated steps to build gstreamer manually.</td>
</tr>
<tr>
<td>v1.5</td>
<td>08 Jan 2016</td>
<td>kstone</td>
<td>Added nvividconv interpolation method.</td>
</tr>
<tr>
<td>v1.5</td>
<td>29 Jan 2016</td>
<td>hlang</td>
<td>Additional syntax changes for 23.2 release</td>
</tr>
<tr>
<td>v2.0</td>
<td>11 May 2016</td>
<td>mzensius</td>
<td>Minor change to nvgstcapture options.</td>
</tr>
<tr>
<td>v3.0</td>
<td>11 Aug 2016</td>
<td>mzensius</td>
<td>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.</td>
</tr>
<tr>
<td>v3.1</td>
<td>06 Oct 2016</td>
<td>mzensius</td>
<td>Minor updates to video encoder features.</td>
</tr>
<tr>
<td>v3.1.1</td>
<td>21 Nov 2016</td>
<td>mzensius</td>
<td>Changed title of document.</td>
</tr>
<tr>
<td>V3.2</td>
<td>12 Jan 2017</td>
<td>mzensius</td>
<td>Adds H.264/H.265 encoder documentation. Also corrects the Gstreamer-1.0 installation procedure</td>
</tr>
<tr>
<td>3.2</td>
<td>03 Mar 2017</td>
<td>hlang</td>
<td>Update date/moniker for L4T 27.1 release. No other updates.</td>
</tr>
<tr>
<td>3.3</td>
<td>13 Jul 2017</td>
<td>mzensius</td>
<td>Minor edit to command syntax, and update of date/moniker for L4T 28.1 release.</td>
</tr>
<tr>
<td>3.4</td>
<td>01 Dec 2017</td>
<td>mzensius</td>
<td>Includes support for NVIDIA® Jetson™ TX1, previously documented elsewhere. Also includes Overlay Sink information, and formatting enhancements.</td>
</tr>
<tr>
<td>3.5</td>
<td>23 Feb 2018</td>
<td>kstone</td>
<td>Added support for the nvarguscamerasrc plugin. Corrected erroneous path. Reformatted commands for line breaks.</td>
</tr>
<tr>
<td>3.5</td>
<td>28 Feb 2018</td>
<td>hlang</td>
<td>Update the Gstreamer installation and setup table to add nvcompositor.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Author</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>3.6</td>
<td>20 April 2018</td>
<td>kstone</td>
<td>Added prerequisites for Video Composition.</td>
</tr>
<tr>
<td>3.7</td>
<td>23 July 2018</td>
<td>jsachs</td>
<td>Add steps to be performed when testing Wayland based GST plugin, playing video, or running Wayland based apps on Wayland display server.</td>
</tr>
<tr>
<td>3.8</td>
<td>29 August 2018</td>
<td>jsachs</td>
<td>Updates for L4T release 31 &amp; GStreamer version 1.14.</td>
</tr>
<tr>
<td>3.9</td>
<td>2 November 2018</td>
<td>jsachs</td>
<td>Updates for L4T release 31.1. Jetson TX1 and TX2 not supported.</td>
</tr>
<tr>
<td>4.0</td>
<td>18 March 2019</td>
<td>jsachs</td>
<td>Updates for L4T release 32.1. Jetson™ TX2 and Jetson Nano™ supported.</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

**Accelerated GStreamer User Guide** ................................................................. 1

GStreamer-1.0 Installation and Setup ................................................................. 1

Decode Examples .............................................................................................. 4
  - Audio Decode Examples Using gst-launch-1.0 ............................................. 4
  - Video Decode Examples Using gst-launch-1.0 ............................................. 5
    - Video Decode Using gst-omx ................................................................. 5
    - Video Decode Using gst-v4l2 ............................................................... 7
  - Image Decode Examples Using gst-launch-1.0 ........................................... 8

Encode Examples ............................................................................................ 8
  - Audio Encode Examples Using gst-launch-1.0 ............................................ 8
  - Video Encode Examples Using gst-launch-1.0 ............................................ 9
    - Video Encode Using gst-omx ............................................................... 9
    - Video Encode Using gst-v4l2 ............................................................ 10
  - Image Encode Examples Using gst-launch-1.0 .......................................... 12

Supported H.264/H.265/VP8/VP9 Encoder Features with GStreamer-1.0 ........... 12
  - Features Supported Using gst-omx ........................................................ 12
  - Features Supported Using gst-v4l2 ........................................................ 17

Camera Capture with GStreamer-1.0 ............................................................... 23

Video Playback with GStreamer-1.0 .............................................................. 24

Video Format Conversion with GStreamer-1.0 ............................................... 27
  - raw-yuv Input Formats ............................................................................ 27
  - raw-yuv Output Formats ....................................................................... 28

NVIDIA Input and Output Formats ................................................................. 28

Video Scaling with GStreamer-1.0 ................................................................. 29
  - raw-yuv Input Formats ............................................................................ 29
  - raw-yuv Output Formats ....................................................................... 30

NVIDIA Input and Output Formats ................................................................. 30

Video Cropping with GStreamer-1.0 .............................................................. 32

Video Transcode with GStreamer-1.0 ............................................................ 32

CUDA Video Post-Processing with GStreamer-1.0 ......................................... 37
  - gst-videocuda ................................................................................... 37
  - gst-nvivafilter .................................................................................. 38

Video Rotation with GStreamer-1.0 ............................................................... 39

Video Composition with GStreamer-1.0 ....................................................... 41

Interpolation Methods for Video Scaling ....................................................... 42

EGLStream Producer Example ....................................................................... 43

EGL Image Transform Example ..................................................................... 43
### GStreamer Build Instructions ............................................................. 45

### Nvgstcapture-1.0 Reference ............................................................ 48
- Command Line Switches ....................................................................... 48
- CSI Camera Supported Resolutions ....................................................... 51
- CSI Camera Runtime Commands .............................................................. 51
- USB Camera Runtime Commands ............................................................ 52
- Notes ...................................................................................................... 53

### nvgstplayer-1.0 Reference .............................................................. 55
- nvgstplayer Command Line Switches ..................................................... 55
- nvgstplayer Runtime Commands ............................................................ 57

### Video Encoder Features ....................................................................... 59

### Supported Cameras ............................................................................. 61
- CSI Cameras .......................................................................................... 61
- USB 2.0 Cameras ................................................................................... 61
- Industrial Camera Details ...................................................................... 61
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 AGX Xavier™ 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:

<table>
<thead>
<tr>
<th>Video Decoder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omxh265dec</td>
<td>OpenMAX IL H.265 Video decoder</td>
</tr>
<tr>
<td>omxh264dec</td>
<td>OpenMAX IL H.264 Video decoder</td>
</tr>
<tr>
<td>omxmpeg4videodec</td>
<td>OpenMAX IL MPEG4 Video decoder</td>
</tr>
<tr>
<td>omxmpeg2videodec</td>
<td>OpenMAX IL MPEG2 Video decoder</td>
</tr>
<tr>
<td>omxvp8dec</td>
<td>OpenMAX IL VP8 Video decoder</td>
</tr>
<tr>
<td>omxvp9dec</td>
<td>OpenMAX IL VP9 video decoder</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Video Encoders</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvv4l2decoder</td>
<td>V4L2 H.265 Video decoder</td>
</tr>
<tr>
<td></td>
<td>V4L2 H.264 Video decoder</td>
</tr>
<tr>
<td></td>
<td>V4L2 VP8 video decoder</td>
</tr>
<tr>
<td></td>
<td>V4L2 VP9 video decoder</td>
</tr>
<tr>
<td></td>
<td>V4L2 MPEG4 video decoder</td>
</tr>
<tr>
<td></td>
<td>V4L2 MPEG2 video decoder</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvv4l2h264enc</td>
<td>V4l2 H.264 video encoder</td>
</tr>
<tr>
<td>nvv4l2h265enc</td>
<td>V4l2 H.265 video encoder</td>
</tr>
<tr>
<td>nvv4l2vp8enc</td>
<td>V4l2 VP8 video encoder (supported with Jetson TX2/TX2i and Jetson Nano; not supported with Jetson AGX Xavier)</td>
</tr>
<tr>
<td>nvv4l2vp9enc</td>
<td>V4l2 VP9 video encoder (supported with Jetson AGX Xavier and Jetson; not supported with Jetson Nano)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Video Sink</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvoverlaysink</td>
<td>OpenMAX IL videosink element</td>
</tr>
</tbody>
</table>

GStreamer version 1.0 includes the following EGL image video sink:

<table>
<thead>
<tr>
<th>Video Sink</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nveglglessink</td>
<td>EGL/GLES videosink element, both the X11 and Wayland backends</td>
</tr>
<tr>
<td>nv3dsink</td>
<td>EGL/GLES videosink element</td>
</tr>
</tbody>
</table>

GStreamer version 1.0 includes the following DRM video sink:

<table>
<thead>
<tr>
<th>Video Sink</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvdrmvideosink</td>
<td>DRM videosink element</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>NVIDIA Proprietary Plugin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvarguscamerasrc</td>
<td>Camera plugin for ARGUS API</td>
</tr>
<tr>
<td>nvvidconv</td>
<td>Video format conversion &amp; scaling</td>
</tr>
<tr>
<td>nvcompositor</td>
<td>Video compositor</td>
</tr>
<tr>
<td>nveglstreamsrc</td>
<td>Acts as GStreamer Source Component, accepts EGLStream from EGLStream producer</td>
</tr>
<tr>
<td>nvvideosink</td>
<td>Video Sink Component. Accepts YUV-I420 format and produces EGLStream (RGBA)</td>
</tr>
<tr>
<td>nvegltransform</td>
<td>Video transform element for NVMM to EGLImage (supported with nveglglessink only)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>JPEG</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvjpegec</td>
<td>JPEG encoder element</td>
</tr>
<tr>
<td>nvjpegdec</td>
<td>JPEG decoder element</td>
</tr>
</tbody>
</table>

**Note**

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

```bash
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)**

```bash
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)**

```bash
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)**

```bash
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)

```text
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:

```text
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)**

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

**H.265 Decode (NVIDIA Accelerated Decode)**

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

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

```text
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)**

```text
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 ! \n '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 ! \n '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 ! omxvpxdec ! \
 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 ! omxvpxdec ! \
 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-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.

```
for example, see the pipeline below.

```

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' ! nvv4l2h264enc \
  maxperf-enable=1 bitrate=8000000 ! h264parse ! qtmux ! filesink \
  location=<filename_h264.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)

```gstreamer
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

```gstreamer
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

```gstreamer
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)

```gstreamer
gst-launch-1.0 videotestsrc num-buffers=200 ! 
  'video/x-raw, width=(int)1280, height=(int)720, 
```
Configuring temporal tradeoff causes the encoder to intentionally, periodically, drop input frames. The following modes are supported:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable</td>
</tr>
<tr>
<td>1</td>
<td>Drop 1 in 5 frames</td>
</tr>
<tr>
<td>2</td>
<td>Drop 1 in 3 frames</td>
</tr>
<tr>
<td>3</td>
<td>Drop 1 in 2 frames</td>
</tr>
<tr>
<td>4</td>
<td>Drop 2 in 3 frames</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disable</td>
</tr>
<tr>
<td>1</td>
<td>Variable bit rate</td>
</tr>
<tr>
<td>2</td>
<td>Constant bit rate</td>
</tr>
<tr>
<td>3</td>
<td>Variable bit rate with frame skip. The encoder skips frames as necessary to meet the target bit rate.</td>
</tr>
<tr>
<td>4</td>
<td>Constant bit rate with frame skip</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UltraFastPreset</td>
</tr>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>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.</td>
</tr>
</tbody>
</table>
Set Profile

gst-launch-1.0 videotestsrc num-buffers=200 ! \ 
'veideo/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:

<table>
<thead>
<tr>
<th>Profile</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Baseline profile</td>
</tr>
<tr>
<td>2</td>
<td>Main profile</td>
</tr>
<tr>
<td>8</td>
<td>High profile</td>
</tr>
</tbody>
</table>

Set Level

gst-launch-1.0 videotestsrc num-buffers=200 is-live=true ! \ 
'veideo/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 ! \ 
'veideo/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 ! \ 
'veideo/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**

```bash
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**

```bash
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:

\[
\text{virtual buffer size} = \text{vbv-size} \times \left(\frac{\text{bitrate}}{\text{fps}}\right)
\]

**Enable Stringent Bitrate**

```bash
gst-launch-1.0 nvarguscamerasrc num-buffers=200 ! \\
   'video/x-raw(memory:NVMM),width=1920,height=1080, \\
   format=(string)NV12' ! \n   omxh264enc control-rate=2 vbv-size=1 EnableTwoPassCBR=true \n   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**

```bash
gst-launch-1.0 videotestsrc num-buffers=200 ! \\
   'video/x-raw, width=(int)1280, height=(int)720, \\
   format=(string)I420' ! \n   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 ! 
  gtmux ! 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)

```bash
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' ! nnv412h264enc \
  control-rate=0 bitrate=30000000 ! h264parse ! qtmux ! filesink \
  location=<filename_h264_VBR.mp4> -e

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' ! nnv412h264enc \
  control-rate=1 bitrate=30000000 ! h264parse ! qtmux ! filesink \
  location=<filename_h264_CBR.mp4> -e
```

The following modes are supported:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Variable bit rate (VBR)</td>
</tr>
<tr>
<td>1</td>
<td>Constant bit rate (CBR)</td>
</tr>
</tbody>
</table>

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

```bash
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' ! nnv412h264enc \
  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)

```bash
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' ! nnv412h264enc \
  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:
<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DisablePreset</td>
</tr>
<tr>
<td>1</td>
<td>UltraFastPreset</td>
</tr>
<tr>
<td>2</td>
<td>FastPreset&lt;br&gt;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.</td>
</tr>
<tr>
<td>3</td>
<td>MediumPreset&lt;br&gt;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.</td>
</tr>
<tr>
<td>4</td>
<td>SlowPreset&lt;br&gt;Supports up to Quarter Pixel (Qpel) block motion estimation. For I/P macroblock mode decisions, intra 4×4 as well as intra 16×16 cost is compared with intermode costs. Supports intra 16×16 and intra 4×4 modes.</td>
</tr>
</tbody>
</table>

Set Profile (Supported with H.264 Encode)

```bash
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:

<table>
<thead>
<tr>
<th>Profile</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Baseline profile</td>
</tr>
<tr>
<td>2</td>
<td>Main profile</td>
</tr>
<tr>
<td>4</td>
<td>High profile</td>
</tr>
</tbody>
</table>

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

```bash
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:

<table>
<thead>
<tr>
<th>Entropy Coding Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CAVLC</td>
</tr>
<tr>
<td>1</td>
<td>CABAC</td>
</tr>
</tbody>
</table>

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_h264.264.264> -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)

```gstreamer
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 quantization range for P, I and B frames.

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

```gstreamer
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)

```gstreamer
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)

```gstreamer
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' ! nv412h265enc ! 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.

<table>
<thead>
<tr>
<th>USB Camera Support</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>YUV</td>
<td>Preview display</td>
</tr>
<tr>
<td></td>
<td>Image capture (VGA, 640 x 480)</td>
</tr>
<tr>
<td></td>
<td>Video capture (480p, 720p, H.264/H.265/VP8/VP9 encode)</td>
</tr>
</tbody>
</table>
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
```

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.
   #                     // window 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.

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

```bash
gst-launch-1.0 filesrc location=<filename_1080p.mp4> ! \
  gtdemux ! h264parse ! omxh264dec \
  nvoverlaysink overlay-x=100 overlay-y=100 overlay-w=640 \
  overlay-h=480 overlay=1 overlay-depth=2 & gst-launch-1.0 filesrc \
  location=<filename_1080p.mp4> ! gtdemux ! 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 `nveglglessink` with the default X11 backend:

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

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

```bash
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 `nveglglesink` 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`:

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

### Properties

`nvdrmvideosink` supports the following properties:

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conn_id</td>
<td>Set connector ID for display.</td>
</tr>
<tr>
<td>plane_id</td>
<td>Set plane ID.</td>
</tr>
<tr>
<td>set_mode</td>
<td>Set default mode (resolution) for playback.</td>
</tr>
</tbody>
</table>

The following command illustrates the use of these properties:

```bash
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`:

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

### 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, BGRx, and RGBA raw-yuv input formats.

- Using the `gst-omx` encoder:
raw-yuv Output Formats

Currently `nvvidconv` supports the I420, UYVY, YUY2, YYVU, NV12, 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:

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

NVIDIA INPUT AND OUTPUT FORMATS

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

<table>
<thead>
<tr>
<th>Input Format</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>NV12</td>
<td>NV12</td>
</tr>
<tr>
<td>I420</td>
<td>I420</td>
</tr>
<tr>
<td>I420_10LE</td>
<td>I420_10LE</td>
</tr>
<tr>
<td>I420_12LE</td>
<td>I420_12LE</td>
</tr>
<tr>
<td>P010_10LE</td>
<td>P010_10LE</td>
</tr>
<tr>
<td>UYVY</td>
<td>UYVY</td>
</tr>
<tr>
<td>YUY2</td>
<td>YUY2</td>
</tr>
</tbody>
</table>
### Input Format

<table>
<thead>
<tr>
<th>Input Format</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>YVYU</td>
<td>YVYU</td>
</tr>
<tr>
<td>BGRx</td>
<td>BGRx</td>
</tr>
<tr>
<td>RGBA</td>
<td>RGBA</td>
</tr>
</tbody>
</table>

Uses these commands to convert between NVIDIA formats:

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

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

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

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
```

## 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, BGRx, and RGBA raw-yuv input formats for scaling.

- Using the `gst-omx` encoder:
Using the **gst-v4l2** encoder:

```c

```nvidia Output Formats

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

Using the **gst-omx** decoder:

```c

```Using the **gst-v4l2** decoder:

```c
```

NVIDIA Input and Output Formats

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

<table>
<thead>
<tr>
<th>Input Format</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>NV12</td>
<td>NV12</td>
</tr>
<tr>
<td>Input Format</td>
<td>Output Format</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>I420</td>
<td>I420</td>
</tr>
<tr>
<td>I420_10LE</td>
<td>I420_10LE</td>
</tr>
<tr>
<td>I420_12LE</td>
<td>I420_12LE</td>
</tr>
<tr>
<td>P010_10LE</td>
<td>P010_10LE</td>
</tr>
<tr>
<td>UYVY</td>
<td>UYVY</td>
</tr>
<tr>
<td>YUY2</td>
<td>YUY2</td>
</tr>
<tr>
<td>YYVU</td>
<td>YYVU</td>
</tr>
<tr>
<td>BGRx</td>
<td>BGRx</td>
</tr>
<tr>
<td>RGBA</td>
<td>RGBA</td>
</tr>
</tbody>
</table>

To scale between NVIDIA formats

- **Using the gst-omx pipeline:**

```
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 pipeline:**

```
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 pipeline:**

```
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 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 !
  ```
Using the `gst-v4l2` pipeline:

```gsh
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:

```gsh
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:

```gsh
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:

```gsh
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:

```gsh
gst-launch-1.0 filesrc location=<filename.webm> ! matroskademux name=demux demux.video_0 ! queue ! nvv4l2decoder !
nvv412h264enc 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 ! \  omxmpeg4videodec ! omxh264enc bitrate=20000000 ! \  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`:

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

- Using the `gst-v4l2` pipeline:

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

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

- Using the `gst-omx-pipeline`:

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

- Using the `gst-v4l2` pipeline:

```gstreamer
gst-launch-1.0 filesrc location=<filename.mp4> ! \
  qtdemux name=demux demux.video_0 ! queue ! h265parse ! \n  nvv4l2decoder ! nvv4l2vp8enc bitrate=20000000 ! queue ! \
  matroskamux name=mux ! \n  filesink location=<Transcoded_filename.mkv> -e
```
VP8 Decode to MPEG-4 Encode (NVIDIA Accelerated Decode to OSS Software Encode)

- Using the `gst-omx-pipeline`:

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

- Using the `gst-v4l2` pipeline:

```bash
gst-launch-1.0 filesrc location=<filename.mkv> ! \n  matroskademux name=demux demux.video_0 ! queue ! nvv4l2decoder ! \n  nvvidconv ! avenc_mpeg4 bitrate=4000000 ! queue ! \n  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`:

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

- Using the `gst-v4l2` pipeline:

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

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

- Using the `gst-omx-pipeline`:

```bash
gst-launch-1.0 filesrc location=<filename.mp4> ! \n  qtdemux name=demux demux.video_0 ! queue ! h264parse ! \n  omxh264dec ! nvvidconv ! theoraenc bitrate=4000000 ! \n  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

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

Sample decode command

```bash
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.

Sample decode pipeline

- Using the `gst-omx-decoder`:

  ```bash
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`:

  ```bash
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

```bash
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.

<table>
<thead>
<tr>
<th>Flip Method</th>
<th>Property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity - no rotation (default)</td>
<td>0</td>
</tr>
<tr>
<td>Counterclockwise - 90 degrees</td>
<td>1</td>
</tr>
<tr>
<td>Rotate - 180 degrees</td>
<td>2</td>
</tr>
<tr>
<td>Clockwise - 90 degrees</td>
<td>3</td>
</tr>
<tr>
<td>Horizontal flip</td>
<td>4</td>
</tr>
<tr>
<td>Upper right diagonal flip</td>
<td>5</td>
</tr>
<tr>
<td>Vertical flip</td>
<td>6</td>
</tr>
<tr>
<td>Upper-left diagonal</td>
<td>7</td>
</tr>
</tbody>
</table>

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:

```bash
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:

```bash
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:

```bash
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`:**

  ```bash
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:**

  ```bash
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 different formats decoded streams

Enter the following command:

```
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::width=1366 sink_2::height=768 \
  sink_3::xpos=0 sink_3::ypos=0 sink_3::width=1280 \
  sink_3::height=720 ! 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
```

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.

<table>
<thead>
<tr>
<th>Interpolation Method</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest</td>
<td>0</td>
</tr>
<tr>
<td>Bilinear</td>
<td>1</td>
</tr>
<tr>
<td>5-tap</td>
<td>2</td>
</tr>
<tr>
<td>10-tap</td>
<td>3</td>
</tr>
<tr>
<td>Smart (default)</td>
<td>4</td>
</tr>
<tr>
<td>Nicest</td>
<td>5</td>
</tr>
</tbody>
</table>

**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:**

```bash
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:**

```bash
gst-launch-1.0 filesrc location=<filename_h264_1080p.mp4>  ! \
  qtdemux ! h264parse ! nvv4l2decoder ! nvegltransform ! 
  nveglglessink -e
```
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-1.15.2 --version=1.15.2
   ```

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-1.15.2/lib/aarch64-linux-gnu
   export PATH=/home/ubuntu/gst-1.15.2/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 1.15.2:
   - gstreamer-1.15.2.tar.xz
   - gst-plugins-base-1.15.2.tar.xz
   - gst-plugins-good-1.15.2.tar.xz
   - gst-plugins-bad-1.15.2.tar.xz
   - gst-plugins-ugly-1.15.2.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_1.15.2/out/lib/pkgconfig

7. Build gstreamer (in this example, gstreamer-1.15.2) with the following commands:

   ./configure --prefix=/home/ubuntu/gst_1.15.2/out
   make
   make install

8. Build gst-plugins-base-1.15.2 with the following commands:

   sudo apt-get install libxv-dev libasound2-dev libtheora-dev \
   libogg-dev libvorbis-dev
   ./configure --prefix=/home/ubuntu/gst_1.15.2/out
   make
   make install

9. Build gst-plugins-good-1.15.2 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_1.15.2/out
10. Obtain and build gst-plugins-bad-1.15.2 with the following commands:

```bash
sudo apt-get install faad libfaad-dev libfaac-dev
./configure --prefix=/home/ubuntu/gst_1.15.2/out
make
make install
```

11. Obtain and build gst-plugins-ugly-1.15.2 with the following commands:

```bash
sudo apt-get install libx264-dev libmad0-dev
./configure --prefix=/home/ubuntu/gst_1.15.2/out
make
make install
```

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

```bash
export LD_LIBRARY_PATH=/home/ubuntu/gst_1.15.2/out/lib/
```

13. Copy the nvidia gstreamer-1.0 libraries to the gst_1.15.2 plugin directory using the following command:

```bash
cd /usr/lib/aarch64-linux-gnu/gstreamer-1.0/
cp libgstnv* libgstromx.so \n~/gst_1.15.2/out/lib/gstreamer-1.0/
```

The nvidia gstreamer-1.0 libraries include:

- `libgstnvarguscamera.so`
- `libgstnvcomposer.so`
- `libgstnvdrmvideosink.so`
- `libgstnvglglessink.so`
- `libgstnvglstreamsrc.so`
- `libgstnvgltransform.so`
- `libgstnvivafilter.so`
- `libgstnvjpeg.so`
- `libgstnvtee.so`
- `libgstnvvidconv.so`
- `libgstnvvideo4linux2.so`
- `libgstnvvideocuda.so`
- `libgstnvvideosink.so`
- `libgstnvvideosinks.so`
- `libgstromx.so`
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 SWITCHES

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

<table>
<thead>
<tr>
<th>“Help” Command Line Switches</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-h</td>
<td>Show help switches.</td>
</tr>
<tr>
<td>--help</td>
<td>Show help switches.</td>
</tr>
<tr>
<td>--help-all</td>
<td>Show all help switches.</td>
</tr>
<tr>
<td>--help-gst</td>
<td>Show GStreamer switches.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Other Command Line Switches</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>--prev_res</td>
<td>Preview width and height.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>--cus-prev-res</td>
<td>Custom preview width and height for CSI only.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>--image_res</td>
<td>Image width and height,</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>--video-res</td>
<td>Video width and height.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>--camsrc</td>
<td>Camera source to use</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-m, --mode</td>
<td>Capture mode.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-v, --video_enc</td>
<td>Video encoder type.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-p, --hw-enc-path</td>
<td>Framework Type.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-b, --enc-bitrate</td>
<td>Video encoding Bit-rate (in</td>
</tr>
<tr>
<td></td>
<td>bytes)</td>
</tr>
<tr>
<td>--enc-controlrate</td>
<td>Video encoding bit-rate control</td>
</tr>
<tr>
<td></td>
<td>method.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>--enc-EnableTwopassCBR</td>
<td>Enable two pass CBR while</td>
</tr>
<tr>
<td></td>
<td>encoding.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>--enc-profile</td>
<td>Video encoder profile (only for</td>
</tr>
<tr>
<td></td>
<td>H.264).</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-j, --image_enc</td>
<td>Image encoder type.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-k, --file_type</td>
<td>Container file type.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>--file-name</td>
<td>Captured file name. nvcamtest is</td>
</tr>
<tr>
<td></td>
<td>used by default.</td>
</tr>
<tr>
<td>--color-format</td>
<td>Color format to use.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>--orientation</td>
<td>Camera sensor orientation value.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>--eglConfig</td>
<td>EGL window Coordinates (x_pos y_pos) in that order.</td>
</tr>
<tr>
<td>-w, --whitebalance</td>
<td>Capture whitebalance value.</td>
</tr>
<tr>
<td>--timeout</td>
<td>Capture timeout value.</td>
</tr>
<tr>
<td>--saturation</td>
<td>Camera saturation value.</td>
</tr>
<tr>
<td>--sensor-id</td>
<td>Camera Sensor ID value.</td>
</tr>
<tr>
<td>--display-id</td>
<td>[For nvooverlaysink only] Display ID value.</td>
</tr>
<tr>
<td>--overlayConfig</td>
<td>Overlay Configuration Options index and coordinates (index, x_pos, y_pos, width, height) order.</td>
</tr>
<tr>
<td>--cap-dev-node</td>
<td>Video capture device node.</td>
</tr>
<tr>
<td>--sys</td>
<td>[For USB] (=) chain for video preview.</td>
</tr>
<tr>
<td>--exposuretimerange</td>
<td>Property to adjust exposure time range in nanoseconds.</td>
</tr>
<tr>
<td>--gainrange</td>
<td>Property to adjust gain range</td>
</tr>
<tr>
<td>--ispdigitalgainrange</td>
<td>Property to adjust digital gain range.</td>
</tr>
<tr>
<td>--aelock</td>
<td>Enable AE Lock.</td>
</tr>
<tr>
<td>--awblock</td>
<td>Enable AWB Lock.</td>
</tr>
<tr>
<td>--exposurecompensation</td>
<td>Property to adjust exposure compensation.</td>
</tr>
<tr>
<td>--aeantibanding</td>
<td>Property to set the auto exposure antibanding mode.</td>
</tr>
<tr>
<td>--tnr-mode</td>
<td>Property to select temporal noise reduction mode.</td>
</tr>
<tr>
<td>--tnr-strength</td>
<td>Property to adjust temporal noise reduction strength.</td>
</tr>
<tr>
<td>--ee-mode</td>
<td>Property to select edge enhancement mode.</td>
</tr>
<tr>
<td>--ee-strength</td>
<td>Property to adjust edge enhancement strength.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Help</td>
<td>—</td>
</tr>
<tr>
<td>q</td>
<td>Quit</td>
<td>—</td>
</tr>
<tr>
<td>mo:&lt;value&gt;</td>
<td>Set capture mode</td>
<td>1-image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-video</td>
</tr>
<tr>
<td>gmo</td>
<td>Get capture mode</td>
<td>—</td>
</tr>
<tr>
<td>so:&lt;val&gt;</td>
<td>Set sensor orientation</td>
<td>0-none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-Rotate counter-clockwise 90 degrees</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-Rotate 180 degrees</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-Rotate clockwise 90 degrees</td>
</tr>
<tr>
<td>gso</td>
<td>Get sensor orientation</td>
<td>—</td>
</tr>
<tr>
<td>wb:&lt;value&gt;</td>
<td>Set white balance mode</td>
<td>0-off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-auto</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2-incandescent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3-fluorescent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4-warm-fluorescent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5-daylight</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6-cloudy-daylight</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7-twilight</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8-shade</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9-manual</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>gwb</td>
<td>Get white balance mode</td>
<td>–</td>
</tr>
<tr>
<td>st:&lt;value&gt;</td>
<td>Set saturation</td>
<td>0-2, e.g., st:1.25</td>
</tr>
<tr>
<td>gst</td>
<td>Get saturation</td>
<td>–</td>
</tr>
<tr>
<td>j</td>
<td>Capture one image.</td>
<td>–</td>
</tr>
<tr>
<td>jx&lt;delay&gt;</td>
<td>Capture after a delay of &lt;delay&gt;, e.g., jx5000 to capture after a 5-second delay</td>
<td>–</td>
</tr>
<tr>
<td>j:&lt;value&gt;</td>
<td>Capture &lt;count&gt; number of images in succession, e.g., j:6 to capture 6 images.</td>
<td>–</td>
</tr>
<tr>
<td>0</td>
<td>Stop recording video</td>
<td>–</td>
</tr>
<tr>
<td>1</td>
<td>Start recording video</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>Video snapshot (while recording video)</td>
<td>–</td>
</tr>
<tr>
<td>gpcr</td>
<td>Get preview resolution</td>
<td>–</td>
</tr>
<tr>
<td>gicr</td>
<td>Get image capture resolution</td>
<td>–</td>
</tr>
<tr>
<td>gvcr</td>
<td>Get video capture resolution</td>
<td>–</td>
</tr>
</tbody>
</table>

**USB CAMERA RUNTIME COMMANDS**

USB Camera Runtime Commands

USB camera runtime commands are described in the following table.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Help</td>
<td>–</td>
</tr>
<tr>
<td>q</td>
<td>Quit</td>
<td>–</td>
</tr>
<tr>
<td>mo:&lt;value&gt;</td>
<td>Set capture mode</td>
<td>1-image 2-video</td>
</tr>
<tr>
<td>gmo</td>
<td>Get capture mode</td>
<td>–</td>
</tr>
<tr>
<td>j</td>
<td>Capture one image.</td>
<td>–</td>
</tr>
<tr>
<td>jx&lt;delay&gt;</td>
<td>Capture after a delay of &lt;delay&gt;, e.g., jx5000 to capture after a 5-second delay</td>
<td>–</td>
</tr>
<tr>
<td>j:&lt;value&gt;</td>
<td>Capture &lt;count&gt; number of images in succession, e.g., j:6 to capture 6 images.</td>
<td>–</td>
</tr>
<tr>
<td>1</td>
<td>Start recording video</td>
<td>–</td>
</tr>
<tr>
<td>0</td>
<td>Stop recording video</td>
<td>–</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>pcr:&lt;value&gt;</td>
<td>Set preview resolution</td>
<td>0-176x144, 1-320x240, 2-640x480, 3-1280x720</td>
</tr>
<tr>
<td>gpcr</td>
<td>Get preview resolution</td>
<td>—</td>
</tr>
<tr>
<td>gicr</td>
<td>Get image capture resolution</td>
<td>—</td>
</tr>
<tr>
<td>gvcr</td>
<td>Get video capture resolution</td>
<td>—</td>
</tr>
<tr>
<td>br:&lt;value&gt;</td>
<td>Set encoding bit rate (in bytes)</td>
<td>e.g., br:4000000</td>
</tr>
<tr>
<td>gbr</td>
<td>Get encoding bit rate</td>
<td>—</td>
</tr>
<tr>
<td>cdn:&lt;value&gt;</td>
<td>Set capture device node</td>
<td>0-/dev/video0, 1-/dev/video1, 2-/dev/video2</td>
</tr>
<tr>
<td>gcdn</td>
<td>Get capture device node</td>
<td>—</td>
</tr>
</tbody>
</table>

**Runtime Video Encoder Configuration Options**

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>br:&lt;val&gt;</td>
<td>Sets encoding bit-rate (in bytes)</td>
<td>Example: br:4000000</td>
</tr>
<tr>
<td>gbr</td>
<td>Gets encoding bit-rate (in bytes)</td>
<td>—</td>
</tr>
<tr>
<td>ep:&lt;val&gt;</td>
<td>Sets encoding profile (for H.264 only)</td>
<td>Example: ep:1 (0): Baseline (1): Main (2): High</td>
</tr>
<tr>
<td>gep</td>
<td>Gets encoding profile (for H.264 only)</td>
<td>—</td>
</tr>
<tr>
<td>Enter ‘f’</td>
<td>Forces IDR frame on video encoder (for H.264 only)</td>
<td>—</td>
</tr>
</tbody>
</table>

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

NVGSTPLAYER COMMAND LINE SWITCHES

Note: To list supported options, enter the command:

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

This table describes nvgstplayer-1.0 command-line switches:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-u &lt;path&gt;</td>
<td>Path of the file containing the URIs.</td>
</tr>
<tr>
<td>--urifile &lt;path&gt;</td>
<td></td>
</tr>
<tr>
<td>-I &lt;uri&gt;</td>
<td>Input URI.</td>
</tr>
<tr>
<td>--uri &lt;uri&gt;</td>
<td></td>
</tr>
<tr>
<td>-e &lt;path&gt;</td>
<td>Element(s) (Properties) file.</td>
</tr>
<tr>
<td>--elemfile &lt;path&gt;</td>
<td></td>
</tr>
<tr>
<td>-x</td>
<td>Command sequence expression.</td>
</tr>
<tr>
<td>--cexpr</td>
<td></td>
</tr>
<tr>
<td>-n &lt;n&gt;</td>
<td>Number of times to play the media.</td>
</tr>
<tr>
<td>--loop &lt;n&gt;</td>
<td></td>
</tr>
<tr>
<td>-c &lt;n&gt;</td>
<td>If a stream has multiple audio tracks, specifies the track number to play.</td>
</tr>
<tr>
<td>--audio-track &lt;n&gt;</td>
<td></td>
</tr>
<tr>
<td>-v &lt;n&gt;</td>
<td>If a stream has multiple video tracks, specifies the track number to play.</td>
</tr>
<tr>
<td>--video-track &lt;n&gt;</td>
<td></td>
</tr>
<tr>
<td>-a &lt;seconds&gt;</td>
<td>Start of the segment in media in seconds.</td>
</tr>
<tr>
<td>--start &lt;seconds&gt;</td>
<td></td>
</tr>
<tr>
<td>Switch</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-d &lt;seconds&gt;</td>
<td>Play duration of the segment in media in seconds.</td>
</tr>
<tr>
<td>--duration &lt;seconds&gt;</td>
<td></td>
</tr>
<tr>
<td>--no-sync</td>
<td>Disable AV sync.</td>
</tr>
<tr>
<td>--disable-dpms</td>
<td>Unconditionally disable DPMS/ScreenBlanking during operation; re-enable on exit.</td>
</tr>
<tr>
<td>--stealth</td>
<td>Operate in stealth mode, staying alive even when no media is playing.</td>
</tr>
<tr>
<td>--bg</td>
<td>Operate in background mode, ignoring keyboard input.</td>
</tr>
<tr>
<td>--use-playbin</td>
<td>Use Playbin.</td>
</tr>
<tr>
<td>--no-audio</td>
<td>Disable audio.</td>
</tr>
<tr>
<td>--no-video</td>
<td>Disable video.</td>
</tr>
<tr>
<td>--disable-anative</td>
<td>Disable native audio rendering.</td>
</tr>
<tr>
<td>--disable-vnative</td>
<td>Disable native video rendering.</td>
</tr>
<tr>
<td>--use-buffering</td>
<td>Use buffering.</td>
</tr>
<tr>
<td>-l &lt;percent&gt;</td>
<td>Low threshold for buffering to start, in percent.</td>
</tr>
<tr>
<td>--low-percent &lt;percent&gt;</td>
<td></td>
</tr>
<tr>
<td>-j &lt;percent&gt;</td>
<td>High threshold for buffering to finish, in percent.</td>
</tr>
<tr>
<td>--high-percent &lt;percent&gt;</td>
<td></td>
</tr>
<tr>
<td>--loop-forever</td>
<td>Play the URI(s) in an endless loop.</td>
</tr>
<tr>
<td>-t &lt;seconds&gt;</td>
<td>Maximum time in queue, in seconds (0=automatic).</td>
</tr>
<tr>
<td>--max-size-time &lt;seconds&gt;</td>
<td></td>
</tr>
<tr>
<td>-y &lt;n&gt;</td>
<td>Maximum amount of memory in the queue, in bytes (0=automatic).</td>
</tr>
<tr>
<td>--max-size-bytes &lt;n&gt;</td>
<td></td>
</tr>
<tr>
<td>-b &lt;n&gt;</td>
<td>Maximum number of buffers in the queue (0=automatic).</td>
</tr>
<tr>
<td>--max-size-buffers &lt;n&gt;</td>
<td></td>
</tr>
<tr>
<td>--window-x &lt;n&gt;</td>
<td>X coordinate for player window (for non-overlay rendering).</td>
</tr>
<tr>
<td>--window-y &lt;n&gt;</td>
<td>Y coordinate for player window (for non-overlay rendering).</td>
</tr>
<tr>
<td>--window-width &lt;n&gt;</td>
<td>Window width (for non-overlay rendering).</td>
</tr>
<tr>
<td>--window-height &lt;n&gt;</td>
<td>Window height (for non-overlay rendering).</td>
</tr>
<tr>
<td>--disable-fullscreen</td>
<td>Play video in non-full-screen mode (for nveglglessink).</td>
</tr>
<tr>
<td>-k &lt;seconds&gt;</td>
<td>Image display time, in seconds.</td>
</tr>
<tr>
<td>--image-display-time &lt;seconds&gt;</td>
<td></td>
</tr>
<tr>
<td>--show-tags</td>
<td>Shows tags (metadata), if available.</td>
</tr>
<tr>
<td>--stats</td>
<td>Shows stream statistics, if enabled.</td>
</tr>
<tr>
<td>--stats-file</td>
<td>File to dump stream statistics, if enabled.</td>
</tr>
<tr>
<td>Switch</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>--svd</td>
<td>(=) chain for video decoding.</td>
</tr>
<tr>
<td>--sad</td>
<td>(=) chain for audio decoding.</td>
</tr>
<tr>
<td>--svc</td>
<td>(=) chain for video postprocessing.</td>
</tr>
<tr>
<td>--sac</td>
<td>(=) chain for audio postprocessing.</td>
</tr>
<tr>
<td>--svs</td>
<td>(=) chain for video rendering.</td>
</tr>
<tr>
<td>--sas</td>
<td>(=) chain for audio rendering.</td>
</tr>
<tr>
<td>--shttp</td>
<td>(=) chain for http source.</td>
</tr>
<tr>
<td>--srtsp</td>
<td>(=) chain for rtsp source.</td>
</tr>
<tr>
<td>--sudp</td>
<td>(=) chain for udp source.</td>
</tr>
<tr>
<td>--sfsrc</td>
<td>(=) chain for file source.</td>
</tr>
</tbody>
</table>

**NVGSTPLAYER RUNTIME COMMANDS**

This table describes nvgstplayer runtime commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>Help</td>
</tr>
<tr>
<td>q</td>
<td>Quit</td>
</tr>
<tr>
<td>Up Arrow</td>
<td>Go to next track.</td>
</tr>
<tr>
<td>Down Arrow</td>
<td>Go to previous track.</td>
</tr>
<tr>
<td>spos</td>
<td>Query for position (time from start).</td>
</tr>
<tr>
<td>sdur</td>
<td>Query for duration.</td>
</tr>
<tr>
<td>s&lt;n&gt;</td>
<td>Seek to &lt;n&gt; seconds from start. e.g. s5.120</td>
</tr>
<tr>
<td>v&lt;pct&gt;</td>
<td>Seek to &lt;pct&gt; percent of the duration. e.g. v54</td>
</tr>
<tr>
<td>f&lt;val&gt;</td>
<td>Shift &lt;n&gt; seconds relative to current position. e.g. f23.901</td>
</tr>
<tr>
<td>Left Arrow</td>
<td>Seek backward 10 seconds.</td>
</tr>
<tr>
<td>Right Arrow</td>
<td>Seek forward 10 seconds.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>p</td>
<td>Pause playback.</td>
</tr>
<tr>
<td>r</td>
<td>Start/resume playback.</td>
</tr>
<tr>
<td>z</td>
<td>Stop playback.</td>
</tr>
<tr>
<td>i:&lt;uri&gt;</td>
<td>Enter a single URI.</td>
</tr>
</tbody>
</table>
The GStreamer-1.0-based `gst-omx` video encoders support the following features, respectively:

<table>
<thead>
<tr>
<th>Video Encoder Feature</th>
<th>H264enc</th>
<th>H265enc</th>
<th>Vp8enc</th>
<th>Vp9enc</th>
</tr>
</thead>
<tbody>
<tr>
<td>profile (Baseline / Main / High)</td>
<td>✓ (all)</td>
<td>✓ (Main)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>level</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>bitrate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>peak bitrate</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>stringent bitrate</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>insert-spspsatidr</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>control-rate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>iframeinterval</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>qp-range</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>temporal-tradeoff</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>bit-packetization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>preset-level</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>low-latency</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>slice-header spacing</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>force-IDR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>vbv-size</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sliceintrarefresenable</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>sliceintrarefreshinterval</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>EnableTwoPassCBR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>num-B-Frames</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
The GStreamer-1.0-based gst-v4l2 video encoders support the following features, respectively:

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

**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


**INDUSTRIAL CAMERA DETAILS**

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

- See3CAM_CU130

- 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.
Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OR CONDITION OF TITLE, MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT, ARE HEREBY EXCLUDED TO THE MAXIMUM EXTENT PERMITTED BY LAW.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA, the NVIDIA logo, CUDA, Jetson, Nano, AGX Xavier, TX2, TX1, and Tegra are trademarks or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright© 2015−2019 NVIDIA Corporation. All rights reserved.