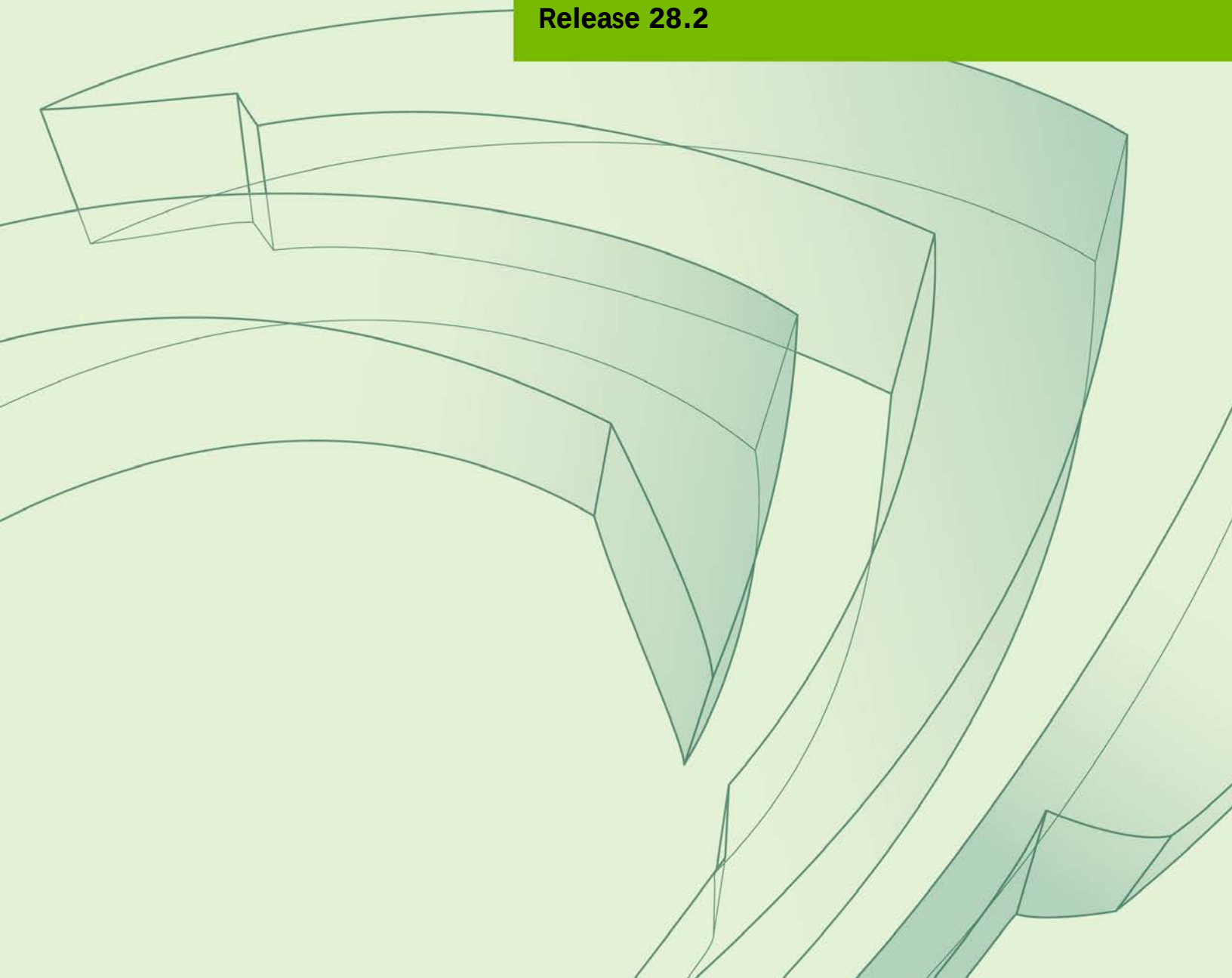
ACCELERATED GSTREAMER USER GUIDE

DA_07303-3.4 | December 1, 2017

Release 28.2



DOCUMENT CHANGE HISTORY

DA_07303

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

TABLE OF CONTENTS

ACCELERATED GSTREAMER USER GUIDE

This document is a user guide for the Gstreamer version 1.0 based accelerated solution included in NVIDIA® Tegra® Linux Driver Package for NVIDIA® Jetson™ TX1 and NVIDIA® Jetson™ TX2 devices.

Note: Any references to Gstreamer version 1.0 apply to Gstreamer version 1.8.3 and prior Gstreamer releases of version 1.x.

This document contains the following sections:

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 version 1.0 includes the following gst-omx video decoders:

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

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

Video Encoders	Description
omxh264enc	OpenMAX IL H.264/AVC video encoder
omxh265enc	OpenMAX IL H.265/AVC video encoder
omxvp8enc	OpenMAX IL VP8 video encoder
omxvp9enc	OpenMAX IL VP9 video encoder (Supported with Jetson TX2)

Gstreamer version 1.0 includes the following gst-omx video sinks:

Video Sink	Description
nvoverlaysink	OpenMAX IL videosink element

Gstreamer version 1.0 includes the following EGL image video sinks:

Video Sink	Description
nveglglessink	EGL/GLES videosink element

Gstreamer version 1.0 includes the following proprietary NVIDIA plugins:

Video Sink	Description
nvvidconv	Video format conversion & scaling
nveglstreamsrc	Acts as Gstreamer Source Component, accepts EGLStream from EGLStream producer
nvvideosink	Video Sink Component. Accepts YUV-I420 format and produces EGLStream (RGBA)
nvegltransform	Video transform element for NVMM to EGLimage (supported with nveglglessink only)

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

Video Sink	Description
nvjpegenc	JPEG encoder element
nvjpegdec	JPEG decoder element

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

export DISPLAY=:0

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

DECODE EXAMPLES

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

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

Audio Decode Examples Using gst-launch-1.0

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

AAC Decode (OSS software decode)

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

AMR-WB Decode (OSS software decode)

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

AMR-NB Decode (OSS software decode)

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

MP3 Decode (OSS software decode)

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

Note: To route audio over HDMI, set the alsasink property device to the following:

`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.

H.264 Decode (NVIDIA accelerated decode)

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

H.265 Decode (NVIDIA accelerated decode)

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

10-bit H265 decode

```
gst-launch-1.0 filesrc location==<filename _10bit.mkv> ! matroskademux
! h265parse ! omxh265dec ! nvvidconv ! 'video/x-raw(memory:NVMM),
format=(string)NV12' ! nvoverlaysink -e
```

Note: Decoding H.265 streams requires Gstreamer version 1.4.x or later, including support for h265parse and qtdemux. See Gstreamer Build Instructions in this guide for details.

VP8 Decode (NVIDIA accelerated decode)

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

Note: When you do not use the primary display to render video, use the `display-id` property of `nvoverlaysink`. For example, refer to the pipeline below.

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

VP9 Decode (NVIDIA accelerated decode)

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

MPEG-4 Decode (NVIDIA accelerated decode)

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

Image Decode

```
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.

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 H265 encode

```
gst-launch-1.0 nvcamerasrc fpsRange="30.0 30.0" ! 'video/x-
raw(memory:NVMM), width=(int)1920, height=(int)1080,
format=(string)I420, framerate=(fraction)30/1' ! nvtee ! nvvidconv !
'video/x-raw(memory:NVMM), format=(string)I420_10LE' ! omxh265enc !
matroskamux ! filesink location=test_10bit.mkv -e
```

VP8 Encode (NVIDIA accelerated encode)

```
gst-launch-1.0 videotestsrc ! 'video/x-raw, format=(string)I420,
width=(int)640, height=(int)480' ! omxvp8enc ! qtmux ! filesink
location=test.mp4 -e
```

VP9 Encode (NVIDIA accelerated, supported with Jetson TX2)

```
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
```

Image Encode

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

Supported H.264/H.265 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 encoder.

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

Set I-frame interval

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

Set temporal-tradeoff (the rate the encoder should drop frames)

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

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

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

Set rate control mode

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

The following modes are supported:

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

Set peak bitrate

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

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

Set quantization range for I, P and B frame

The format for the range is the following:

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

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

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

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

Set hardware preset level

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

The following modes are supported:

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

Set profile

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

From `omxh264enc`, the following profiles are supported:

Profile	Description
1	Baseline profile
2	Main profile
8	High profile

Set level

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

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

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

Set number of B frames between two reference frames

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

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

Insert SPS and PPS at IDR

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

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

Enable two-pass CBR

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

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

Set virtual buffer size

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

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

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

Enable stringent bitrate

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

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

Slice-header-spacing with spacing in terms of MB

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

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

Slice header spacing with spacing in terms of number of bits

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

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

CAMERA CAPTURE WITH GSTREAMER-1.0

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

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

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

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

The following table shows USB camera support.

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

raw-yuv Capture (I420 format) and preview display with xvimagesink

```
gst-launch-1.0 v4l2src device="/dev/video0" ! "video/x-raw, width=640,
height=480, format=(string)I420" ! 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:

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)

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

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

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

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

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

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

VIDEO FORMAT CONVERSION WITH GSTREAMER-1.0

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

raw-yuv Input Formats

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

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

raw-gray Input Formats

Currently `nvvidconv` supports the GRAY8 raw-gray input format.

```
gst-launch-1.0 videotestsrc ! 'video/x-raw, format=(string)GRAY8,
width=(int)1280, height=(int)720’ ! nvvidconv !
'video/x-raw(memory:NVMM), format=(string)I420’ ! omxh264enc !
'video/x-h264,
stream-format=(string)byte-stream' ! h264parse ! qtmux ! filesink
location=test.mp4 -e
```

raw-yuv Output Formats

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

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

raw-gray Output Formats

Currently nvvidconv supports the GRAY8 raw-gray output format.

```
gst-launch-1.0 filesrc location=640x480_30p.mp4 ! qtdemux ! queue !
h264parse ! omxh264dec ! nvvidconv ! 'video/x-raw,
format=(string)GRAY8' ! videoconvert ! xvimagesink -e
```

VIDEO SCALING WITH GSTREAMER-1.0

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

raw-yuv Input Formats

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

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

raw-gray Input Formats

Currently nvvidconv supports the GRAY8 raw-gray input format for scaling.

```
gst-launch-1.0 videotestsrc ! 'video/x-raw, format=(string)GRAY8,
width=(int)1280, height=(int)720’! nvvidconv !
'video/x-raw(memory:NVMM), width=(int)640, height=(int)480,
format=(string)I420' !  omxh264enc ! 'video/x-h264, stream-
```

```
format=(string)byte-stream' ! h264parse ! qtmux ! filesink
location=test.mp4 -e
```

raw-yuv Output Formats

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

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

raw-gray Output Formats

Currently `nvvidconv` supports the GRAY8 raw-gray output format for scaling.

```
gst-launch-1.0 filesrc location=1280x720_30p.mp4 ! qtdemux ! queue !
h264parse ! omxh264dec ! nvvidconv ! 'video/x-raw,
format=(string)GRAY8, width=640, height=480' ! videoconvert !
xvimagesink -e
```

NVIDIA Input and Output Formats

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

Input Format	Output Format
NV12	NV12
I420, I420_10LE	I420, I420_10LE
	RGBA

To scale between NVIDIA formats

- Scale between NVIDIA Formats with the following commands:

```
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)I420' ! omxh264enc ! qtmux ! filesink
location=test.mp4 -e

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

gst-launch-1.0 nvcamerasrc fpsRange="30 30" ! 'video/x-
raw(memory:NVMM), width=(int)1920, height=(int)1080,
format=(string)I420, framerate=(fraction)30/1' ! nvtee ! nvvidconv !
'video/x-raw(memory:NVMM), width=(int)640, height=(int)480,
format=(string)NV12’ ! omxh264enc ! qtmux ! filesink location=test.mp4
-e
```

VIDEO CROPPING WITH GSTREAMER-1.0

The NVIDIA proprietary nvvidconv Gstreamer-1.0 plug-in also allows you to perform video cropping.

To crop video

- Crop video with the following commands:

```
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
```

VIDEO TRANSCODE WITH GSTREAMER-1.0

You can perform video transcoding between the following video formats.

H.264 Decode to VP8 Encode (NVIDIA-accelerated decode to NVIDIA-accelerated encode)

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

H.265 Decode to VP9 Encode (NVIDIA-accelerated decode to NVIDIA-accelerated encode)

```
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
```

VP8 Decode to H.264 Encode (NVIDIA-accelerated decode to NVIDIA-accelerated encode)

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

VP9 Decode to H.265 Encode (NVIDIA-accelerated decode to NVIDIA-accelerated encode)

```
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
```

MPEG-4 Decode to VP8 Encode (NVIDIA-accelerated decode to NVIDIA-accelerated encode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! qtdemux name=demux
demux.video_0 ! queue ! mpeg4videoparse ! omxmpeg4videodec ! omxvp8enc
bitrate=20000000 ! qtmux name=mux ! filesink
location=<Transcoded_filename.mp4> -e
```

MPEG-4 Decode to H.264 Encode (NVIDIA-accelerated decode to NVIDIA-accelerated encode)

```
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 MPEG-4 Encode (NVIDIA-accelerated decode to OSS software encode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! qtdemux name=demux
demux.video_0 ! queue ! h264parse ! omxh264dec ! nvvidconv !
avenc_mpeg4 bitrate=4000000 ! qtmux name=mux ! filesink
location=<Transcoded_filename.mp4> -e
```

H.265 Decode to MPEG-4 Encode (NVIDIA-accelerated decode to OSS software encode)

```
gst-launch-1.0 filesrc location=<filename.mp4> ! qtdemux name=demux
demux.video_0 ! queue ! h265parse ! omxh265dec ! nvvidconv !
avenc_mpeg4 bitrate=4000000 ! qtmux name=mux ! filesink
location=<Transcoded_filename.mp4> -e
```

VP8 Decode to MPEG-4 Encode (NVIDIA-accelerated decode to OSS software encode)

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

VP9 Decode to MPEG-4 Encode (NVIDIA-accelerated decode to OSS software encode)

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

H.264 Decode to Theora Encode (NVIDIA-accelerated decode to OSS software encode)

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

H.264 Decode to H.263 Encode (NVIDIA-accelerated decode to OSS software encode)

```
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
```

CUDA VIDEO POST-PROCESSING WITH GSTREAMER-1.0

This section describes Gstreamer-1.0 plug-ins for NVIDIA® CUDA® post-processing operations.

gst-videocuda

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

The following are sample pipeline creation and application usage commands.

Sample decode pipeline

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

Sample decode command

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

gst-nvivafilter

This NVIDIA proprietary GStreamer-1.0 plug-in 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

```
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
```

Sample CSI Camera pipeline

```
gst-launch-1.0 nvcamerasrc fpsRange="30 30" ! 'video/x-
raw(memory:NVMM), width=(int)3840, height=(int)2160,
format=(string)I420, framerate=(fraction)30/1' ! nvtee ! nvivafilter
cuda-process=true customer-lib-name="libnvsample_cudaprocess.so" !
'video/x-raw(memory:NVMM), format=(string)NV12' ! nvoverlaysink -e
```

Note: See 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 plug-in also allows you to perform video rotation operations.

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

Flip Method	Property value
identity - no rotation (default)	0
counterclockwise - 90 degrees	1
rotate - 180 degrees	2
clockwise - 90 degrees	3
horizontal flip	4
upper right diagonal flip	5
vertical flip	6
upper-left diagonal	7

Note: Get information on 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.

```
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
```

To rotate video 90 degrees clockwise

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

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

To rotate 180 degrees

- To rotate video 180 degrees, enter the following command:

```
gst-launch-1.0 nvcamerasrc fpsRange="30.0 30.0" ! 'video/x-
raw(memory:NVMM), width=(int)1920, height=(int)1080,
format=(string)I420, framerate=(fraction)30/1' ! nvtee ! 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:

```
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
```

To scale and rotate video 90 degrees clockwise

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

```
gst-launch-1.0 nvcamerasrc fpsRange="30.0 30.0" ! 'video/x-
raw(memory:NVMM), width=(int)1920, height=(int)1080,
format=(string)I420, framerate=(fraction)30/1' ! nvtee ! 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:

```
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
```

Note: Currently, nvcompositor only supports video decode(gst-omx) with overlay render pipeline.

INTERPOLATION METHODS FOR VIDEO SCALING

The NVIDIA proprietary `nvvidconv` Gstreamer-1.0 plug-in allows you to choose the interpolation method used for scaling.

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

Interpolation Method	Property Value
nearest	0
bilinear	1
5-Tap	2
10-Tap	3
smart (default)	4
Nicest	5

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

To use bilinear interpolation method for scaling

- Enter the following command:

```
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
```

EGLSTREAM PRODUCER EXAMPLE

The NVIDIA-proprietary nveglstreamsrc and nvvideosink Gstreamer-1.0 plug-ins 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 --nvvideosink-create-eglstream
```

EGL IMAGE TRANSFORM EXAMPLE

The NVIDIA proprietary nvegltransform Gstreamer-1.0 plug-in allows simulation of an EGLImage transform pipeline.

To simulate an EGL Image transform pipeline

- Enter the following command:

```
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
```

GSTREAMER BUILD INSTRUCTIONS

This release contains the `git-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.gst.12.3 --version=1.12.3
```

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-1.12.3/lib/aarch64-linux-gnu
export PATH==/home/ubuntu/gst-1.12.3/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 1.12.3:

- gstreamer-1.12.3.tar.xz
- gst-plugins-base-1.12.3.tar.xz
- gst-plugins-good-1.12.3.tar.xz
- gst-plugins-bad-1.12.3.tar.xz
- gst-plugins-ugly-1.12.3.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 with the following command:

```
export PKG_CONFIG_PATH=/home/ubuntu/gst_1.12.3/out/lib/pkgconfig
```

7. Build gstreamer (in this example, gstreamer-1.12.3) with the following commands:

```
./configure --prefix=/home/ubuntu/gst_1.12.3/out
make
make install
```

8. Build gst-plugins-base-1.12.3 with the following commands:

```
sudo apt-get install libxv-dev libasound2-dev libtheora-dev libogg-
dev libvorbis-dev
./configure --prefix=/home/ubuntu/gst_1.12.3/out
make
make install
```

9. Build gst-plugins-good-1.12.3 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_1.12.3/out
make
make install
```

10. Obtain and build gst-plugins-bad-1.12.3 with the following commands:

```
sudo apt-get install faad libfaad-dev libfaac-dev
./configure --prefix=/home/ubuntu/gst_1.12.3/out
make
make install
```

11. Obtain and build gst-plugins-ugly-1.12.3 with the following commands:

```
sudo apt-get install libx264-dev libmad0-dev
./configure --prefix=/home/ubuntu/gst_1.12.3/out
make
make install
```

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

```
export LD_LIBRARY_PATH=/home/ubuntu/gst_1.12.3/out/lib/
```

13. Copy the `nvidia gstreamer-1.0` libraries to the `gst_1.12.3` plugin directory using the following command:

```
cd /usr/lib/arm-linux-gnueabihf/gstreamer-1.0/
cp libgstnv* libnvgst* libgstomx.so ~/gst_1.12.3/out/lib/gstreamer-
1.0/
```

The `nvidia gstreamer-1.0` libraries include:

```
libgstnvcamera.so
libgstnveglglessink.so
libgstnveglstreamsrc.so
libgstnvegltransform.so
libgstnvivafilter.so
libgstnvvidconv.so
libgstnvvideosink.so
libnvgstjpeg.so
libgstomx.so
```

NVGSTCAPTURE-1.0 OPTION REFERENCE

This section describes the options available in the `nvgstcapture-1.0` application.

NVGSTCAPTURE APPLICATION OPTIONS

Nvgstcapture-1.0 command-line options are described in the following table.

Application Options		
Option	**Description**	**Notes**
--prev_res	Preview area width and height, e.g., --prev_res=3	-
--cus-prev-res	Custom preview width and height for CSI only	-
--image_res	Image width and height, e.g., --image_res=3	-
--video_res	Video width and height, e.g., --video_res=3	-
-m, --mode	Capture mode.	1-Still 2-Video
--color-format	Color format type	0-I420[default for V4L2] 1-NV12[For CSI only] 2-YUY2[For V4L2 only]
-v, --video_enc	Video encoder type.	0-H.264 (hardware) 1-VP8(hardware) 2-MPEG-4 (software) 3-H.263 (software)
-b, --enc-bitrate	Video encoding Bit-rate (in bytes)	Example: --enc-bitrate=4000000
--enc-profile	Video encoder profile (only for H.264)	0-Baseline 1-Main 2-High

-j, --image_enc	Image encoder type.	0-jpeg_SW[jpegenc] 1-jpeg_HW[nvjpegenc]
-k, --file_type	Container file type.	0-MP4 1-3GP 2-AVI
--cap-dev-node	Video capture device node.	0=/dev/video0[default] 1=/dev/video1 2=/dev/video2
--svs	Chain for video preview.	-
--file-name	File name for capture.	“nvcamtest” is used by default.
--camsrc	Camera source.	0-v4l2 1-csi (default) 2-videotest 3-eglstream
--nvvideosink-create-eglstream	Enable nvvideosink EGLstream Producer(nvvideosink EGLStream Producer only)	-
--aeLock	Enable auto exposure lock(CSI only)	-
--orientation	Camera sensor orientation value(CSI only)	-
-w, --whitebalance	White balance value for capture. (CSI only)	-
-s, --scene-mode	Camera scene-mode value. (CSI only)	-
-c, --color-effect	Camera color effect value. (CSI only)	-
--auto-exposure	Camera auto-exposure value. (CSI only)	-
--flash	Camera flash value. (CSI only)	-
--flicker	Camera flicker detection and avoidance mode value. (CSI only)	-
--contrast	Camera contrast value. (CSI only)	-
--saturation	Camera saturation value. (CSI only)	-
--edge-enhancement	Camera edge enhancement value. (CSI only)	-
--tnr_strength	Camera TNR strength value. (CSI only)	-
--tnr_mode	Camera TNR mode value. (CSI only)	-
--sensor-id	Camera Sensor ID value. (CSI only)	-
--display-id	Display ID value (for nvoverlaysink only)	-

--eglstream-id	Select EGLStreamProducerID value (for CSI EGLStream). Default is 0.	-
--aeRegion	ROI for AE coordinates (top, left, bottom, right) and weight, in that order. (CSI only)	Example: --aeRegion="30 40 200 200 1.2"
--wbRegion	ROI for AWB coordinates (top, left, bottom, right) and weight in that order. (CSI only)	Example: --wbRegion="30 40 200 200 1.2"
--fpsRange	FPS range values (low, high) (CSI only)	Example: --fpsRange="15 30"
--wbGains	White Balance (WB) gains values (R, GR, GB, B) in that order. (CSI only)	Example: --wbGains="1.2 1.4 0.8 1.6"
--overlayConfig	Overlay Configuration Options index and coordinates in (index, x_pos, y_pos, width, height) order.	Example: --overlayConfig="0, 0, 0, 1280, 720"
--enable-meta	Enables Sensor MetaData reporting if the sensor has the capability to provide the embedded metadata.	-
--eglConfig	EGL window Coordinates (x_pos y_pos) in that order.	Example: --eglConfig=”50 100”
--enable-exif	Enable Exif data	-
--dump-bayer	Dump bayer data in addition to image capture	-
--exposure-time	Capture exposure time value. (CSI only)	Example: --exposure-time=0.033
Help Options		
Option	**Description**	**Notes**
-h, --help	Show help options.	-
--help-all	Show all help options.	-
--help-gst	Show Gstreamer options.	-

CSI CAMERA SUPPORTED RESOLUTIONS

CSI camera supports the following image resolutions:

- 640x480
- 1280x720
- 1920x1080
- 2104x1560
- 2592x1944
- 2616x1472
- 3840x2160

- 3896x2192
- 4208x3120
- 5632x3168
- 5632x4224

CSI CAMERA RUNTIME COMMANDS

CSI camera runtime commands are described in the following table.

Command	Description	Notes
h	Help	-
q	Quit	-
mo:<value>	Set capture mode	1-image 2-video
gmo	Get capture mode	-
sid:<value>	Set sensor ID	-
gsid	Get sensor ID	-
so:<val>	Set sensor orientation	0-none 1-Rotate counter-clockwise 90 degrees 2-Rotate 180 degrees 3-Rotate clockwise 90 degrees
gso	Get sensor orientation	-
wb:<value>	Set white balance mode	0-off 1-auto 2-incandescent 3-fluorescent 4-warm-fluorescent 5-daylight 6-cloudy-daylight 7-twilight 8-shade
gwb	Get white balance mode	-

scm:<value>	Set scene mode	0-face-priority 1-action 2-portrait 3-landscape 4-night 5-night-portrait 6-theatre 7-beach 8-snow 9-sunset 10-steady-photo 11-fireworks 12-sports 13-party 14-candle-light 15-barcode
gscm	Get scene mode	-
ce:<value>	Set color effect mode	1-off 2-mono 3-negative 4-solarize 5-sepia 6-posterize 7-aqua
gce	Get color effect mode	-
ae:<value>	Set auto-exposure mode	1-off 2-on 3-OnAutoFlash 4-OnAlwaysFlash 5-OnFlashRedEye
gae	Get auto exposure mode	-
ael	Set auto-exposure lock	0-unlock 1-lock
gael	Get auto-exposure lock	-
f:<value>	Set flash mode	0-off 1-on 2-torch 3-auto
gf	Get flash mode	-
fl:<value>	Set flicker detection and avoidance mode	0-off 1-50 Hz 2-60 Hz 3-auto
gfl	Get flicker detection and avoidance mode	-
ct:<value>	Set contrast	0-1, e.g., ct:0.75
gct	Get contrast	-
st:<value>	Set saturation	0-2, e.g., st:1.25
gst	Get saturation	-

ext:<value>	Set exposure time (in seconds)	e.g., ext:0.033
gext	Get exposure time	-
ee:<value>	Set edge enhancment	0-1, e.g., ee:0.75
gee	Get edge enhancment	-
aer:<value>	Set ROI coordinates for AE (top, left, bottom, right) and weight	e.g., aer:20 20 400 400 1.2
gaer	Get ROI for AE	-
wbr:<value>	Set ROI coordinates for AWB (top, left, bottom, right) and weight	e.g., wbr:20 20 400 400 1.2
gwbr	Get ROI for AE	-
fpsr:<value>	Set FPS range (low, high)	e.g., fpsr:15 30
gfpsr	Get FPS range	-
wbg:<value>	Set WB gains (R, GR, GB, B)	e.g., wbg:1.2 2.2 0.8 1.6
gwbg	Get WB gains	-
ts:<value>	Set TNR strength	0-1, e.g., ts:0.75
gts	Get TNR strength	-
tnr:<value>	Set TNR mode	0-Original 1-Outdoor-low-light 2-Outdoor-medium-light 3-Outdoor-high-light 4-Indoor-low-light 5-Indoor-medium-light 6-Indoor-high-light
gtnr	Get TNR mode	-
j	Capture one image.	-
jx<delay>	Capture after a delay of <delay>, e.g., jx5000 to capture after a 5-second delay	-
j:<value>	Capture <count> number of images in succession, e.g., j:6 to capture 6 images.	-
1	Start recording video	-
0	Stop recording video	-
pcr	Set preview resolution	pcr:<0..11>
gpcr	Get preview resolution	-
icr	Set image capture resolution	icr:<0..12>
gicr	Get image capture resolution	-
vcr	Set video capture resolution	vcr:<0..9>
gvcr	Get video capture resolution	-

USB CAMERA RUNTIME COMMANDS

USB camera runtime commands are described in the following table.

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

Runtime video encoder configuration options are described in the following table.

Command	Description	Notes
br:<val>	Sets encoding bit-rate (in bytes)	Example: br:4000000
gbr	Gets encoding bit-rate (in bytes)	-
ep:<val>	Sets encoding profile (for H.264 only)	Example: ep:1 (0): Baseline

		(1): Main (2): High
gep	Gets encoding profile (for H.264 only)	-
Enter+f	Forces IDR frame on video encoder (for H.264 only)	-

NOTES

- The nvgstcapture-1.0 application generates image and video output files in the same directory as the application itself.
- Filenames for image and video content are in the formats `nvcamtest<counter>.jpg` and `nvcamtest<counter>.mp4` respectively, where <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.
- Default H.263 encode resolution is 704x576(4CIF) in AVI container formats. Use `--camsrc=2` for H.263 video encode.
- 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.

VIDEO ENCODER FEATURES

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

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

SUPPORTED CAMERAS

This section describes the supported cameras.

CSI CAMERAS

- Jetson TX1 and Jetson TX2 currently support 2 CSI RAW BAYER sensors.
- The platform has been validated with a single OV5693 sensor and a single IMX185 for capture on L4T.
- The camera module is interfaced with the Tegra platform via MIPI-CSI.
- Tested using the nvgstcapture application.

USB 2.0 CAMERAS

The following cameras have been validated on Tegra platforms for Android and L4T with USB 2.0 ports. These cameras are UVC compliant.

- Logitech c920 (preferred)

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

- Logitech c910

  http://www.amazon.com/Logitech-HD-Pro-Webcam-C910/dp/B003M2YT96

- Rocketfish™ HD Webcam Pro

  http://www.rocketfishproducts.com/products/computer-accessories/RF-HDWEB10.html?supportTab=open

- Creative Live! Cam Socialize HD 1080

  http://support.creative.com/Products/ProductDetails.aspx?catID=218&CatName=Web+Cameras&subCatID=231&subCatName=MIDI+Keyboards&prodID=20165&prodN

ame=Live!+Cam+Socialize+HD+1080&bTopTwenty=1&VARSET=prodfaq:PRODFAQ_20165,VARSET=CategoryID:218

INDUSTRIAL CAMERA DETAILS

The following USB 3.0 Industrial cameras are supported on Jetson-TX1 under L4T:

- See3CAM_CU130

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

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

- MQ003CG-CM

  http://www.ximea.com/en/products/usb3-vision-cameras-xiq-line/mq003cg-cm

  - USB 3.0
  - Non-UVC compliant
  - 640 x 480 at 500 FPS
  - Purpose - Embedded Robotics
  - Installation and Verification on Jetson TX1:

    1. Add the user to the plugdev group:

       ```
       sudo gpasswd -a ubuntu plugdev
       ```

       Re-login.

    2. Install tools for the application:

       ```
       apt-get install libgstreamer0.10-dev libgstreamer-plugins-
       base0.10-dev libgtk2.0-dev g++"
       ```

    3. Download XIMEA Linux Software Package:

       ```
       wget http://www.ximea.com/downloads/recent/XIMEA_Linux_SP.tgz
       ```

       Untar:

```
tar xzf XIMEA_Linux_SP.tgz
cd package
```

4. Open the install file and replace

```
elif [ "${arch:0:3}" == "arm" ]
```

with

```
elif [ "$arch" == "aarch64" ]
```

5. Start installation:

```
./install
```

Install USB3 camera:

```
./install -cam_usb30
```

Install graphical desktop:

```
sudo apt-get update
sudo apt-get install ubuntu-desktop
```

6. Reboot. The system boots to the graphical desktop.
7. To access sample applications:
   - xiSample: run from `/package/bin` folder
   - streamViewer
     - `make` from `/package/examples/streamViewer` folder
     - run from the `/package/bin` folder

Notice

Trademarks

Copyright