Qualcomm Intelligent Multimedia SDK (QIM SDK) Reference

80-50450-50 Rev. AC

December 20, 2023

Revision history

Revision	Date	Description
AC	December 2023	Added the following: ▪ gst-camera-burst-intervalcapture-example ▪ gst-tflite-yolo-ssd-display-example
AB	October 2023	▪ Updated release details for QIM SDK versions in Table 1-1 ▪ Added the TensorFlow Lite use cases and Qualcomm Neural Processing SDK use cases: Machine learning use cases ▪ Added a section that demonstrates running of the TFLite PoseNet model: gst-tflite-posenet-display-example
AA	August 2023	Initial release

Contents

Figures

Tables

1 Introduction

The Qualcomm® intelligent multimedia software development kit (QIM SDK 1.0.0) is a unified SDK across Internet of Things (IOT) segments enabling seamless multimedia and artificial intelligence/ machine learning (AI/ML) application deployment. This SDK uses GStreamer, an open-source multimedia framework and exposes easy APIs and plug-ins in both multimedia and ML domain.

These plug-ins enable application developers to develop various multimedia and AI/ML applications across various segments such as smart/connected/IP/Sports/Web cameras, Robotics/Drones, AI, or ML box and so on.

The application can be single/multi stream multimedia or ML or combination of both. This SDK enables flexible construction of use cases/pipelines by providing a catalog of Qualcomm plug-ins with hardware acceleration. These plug-ins are optimized to run on Qualcomm hardware/IPs and enable the end application to run in the most efficient way. The plug-ins constitute video encode/decode, Camera ISP, GPU, display, audio DSP (aDSP), compute DSP (cDSP), and AI/ML accelerators.

The plug-in style framework allows the developer to plug and play with these plug-ins to build specific multimedia or AI/ML applications with ease of use and flexibility. The developer does not need to understand the low-level platform libraries and hardware details, which can vary across chipsets. This SDK hides the complexity of hardware within the plug-in architecture and provides easy APIs to applications. The applications built on top of this SDK can run seamlessly in different version or different tiers of the chipset.

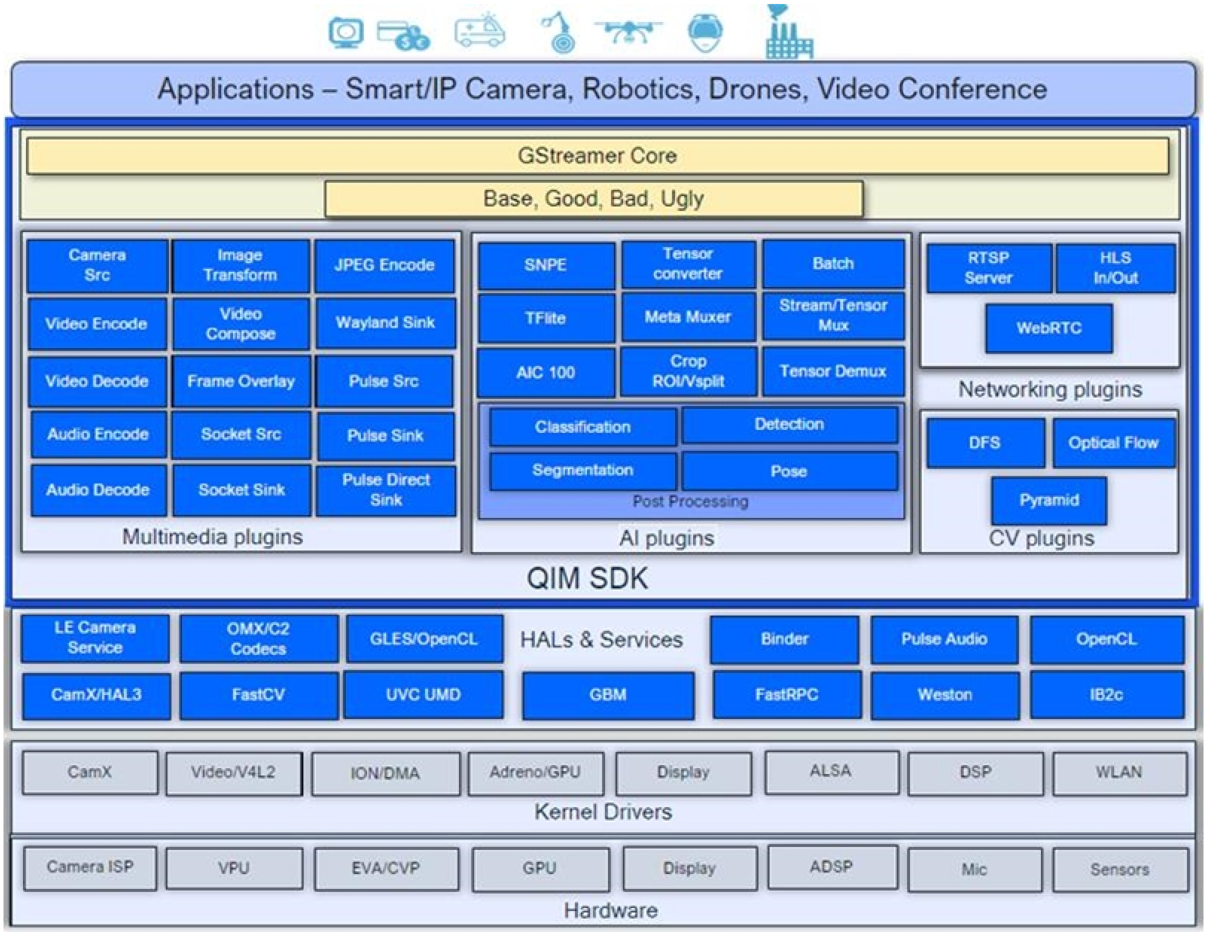


Figure 1-1 IMSDK GStreamer plug-in architecture

To set up the QIM SDK build and development environment, see *Qualcomm® Intelligent Multimedia SDK (QIM SDK) Quick Start Guide* (80-50450-51).

Table 1-1 Release information

QIM SDK version	CodeLinaro release tag
V1.0.1	QIM.SDK.1.0.0.r1-01100-QIM.0
V1.0	TFLITE.SDK.1.0.r1-00200-TFLITE.0

References

Table 1-2 Related documents

Title	Number
Qualcomm	
Qualcomm Intelligent Multimedia SDK (QIM SDK) Quick Start Guide	80-50450-51
Qualcomm TensorFlow Lite SDK Tools Quick Start Guide	80-50450-52

2 Multimedia architecture

The GStreamer plugins interact with various multimedia components, which enables you implement various use cases.

2.1 Camera

The camera plug-in of QIM SDK 1.0.0 is qtiqmmfsrc. The plug-in acts as a client to the QTI Camera Service.

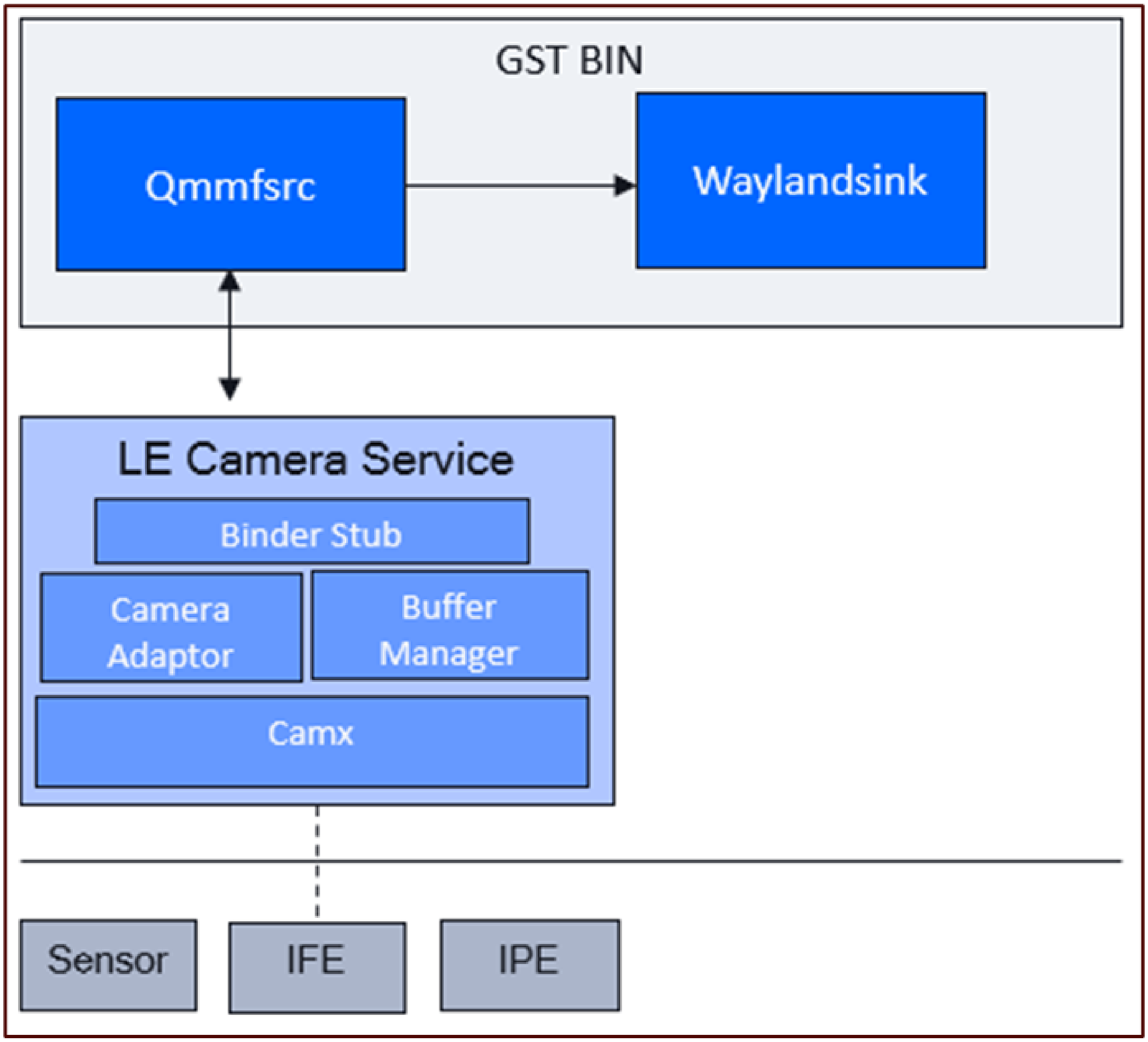


Figure 2-1 Camera architecture

- The Camera Service runs as a daemon in the system and provides easy remote procedure call (RPC) APIs to control the camera. It exposes helper client APIs, which perform the binder RPC between client and server.
- The client-server architecture enables the multiclient and multicamera use cases to do the following:
  - Create many instances of the qtiqmmfsrc camera plug-in where one instance of the plug-in corresponds to one physical or logical camera
  - Implement camera use cases. These instances can be in the same process or different processes depending upon the use case

- The qtiqmmfsrc camera plug-in provides multiple streams in parallel (multiple source pads), and each stream can be of different formats (NV12/21 or MJPEG).

  For example, if there are three parallel streams as listed:

  a. One stream for local storage

  b. One stream for YUV stream for live camera preview on a local display

  c. One for network streaming

  The plug-in can support ‘N’ number of parallel streams, where 'N' depends on camera image signal processor (ISP) capability on a specific QTI chipset.

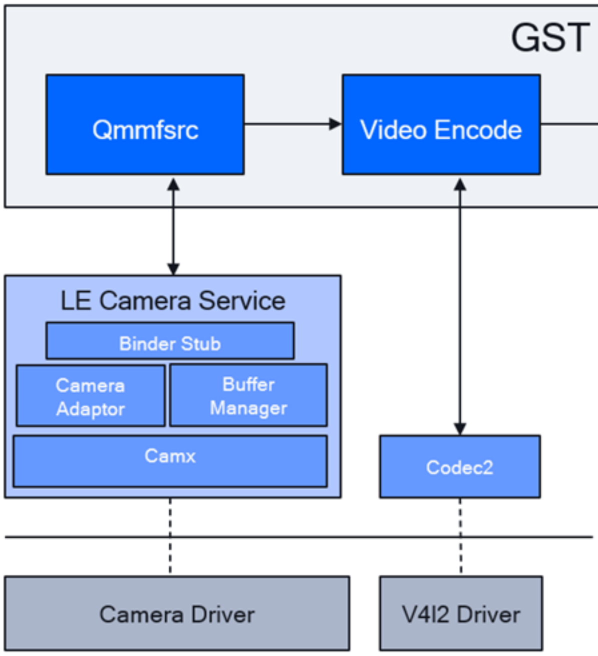
- The waylandsink element uses the raw YUV (NV12/21) stream output from the plug-in to render the camera frames to a physical display. This can help with the following:
  - Achieve a live camera preview use case
  - Enable the postprocessing element to use it to improve the quality
  - Enable the ML inferencing elements to use it to do inferencing on live camera
- The Camera Service uses the HAL3 API, which interacts with the camera back-end (CamX) and camera driver to configure camera sensor and ISP hardware.
- A GBM is a common platform library to allocate graphics/image buffers. The Camera Service uses GBM to allocate buffers for each camera stream and submits them to HAL3
- The same buffers are circulated to the clients (qtiqmmfsrc) and other plug-ins in the pipeline with zero copy
- The buffer pools manage the buffers to avoid per frame allocation

2.2 Video

The video use cases comprise video encode using the qtic2venc plug-in, and video decode using the qtic2vdec plug-in.

2.2.1 Video encode

The video encode plug-in (qtic2venc) connects to the camera plug-in to implement video encode (H264 or H265) use cases.



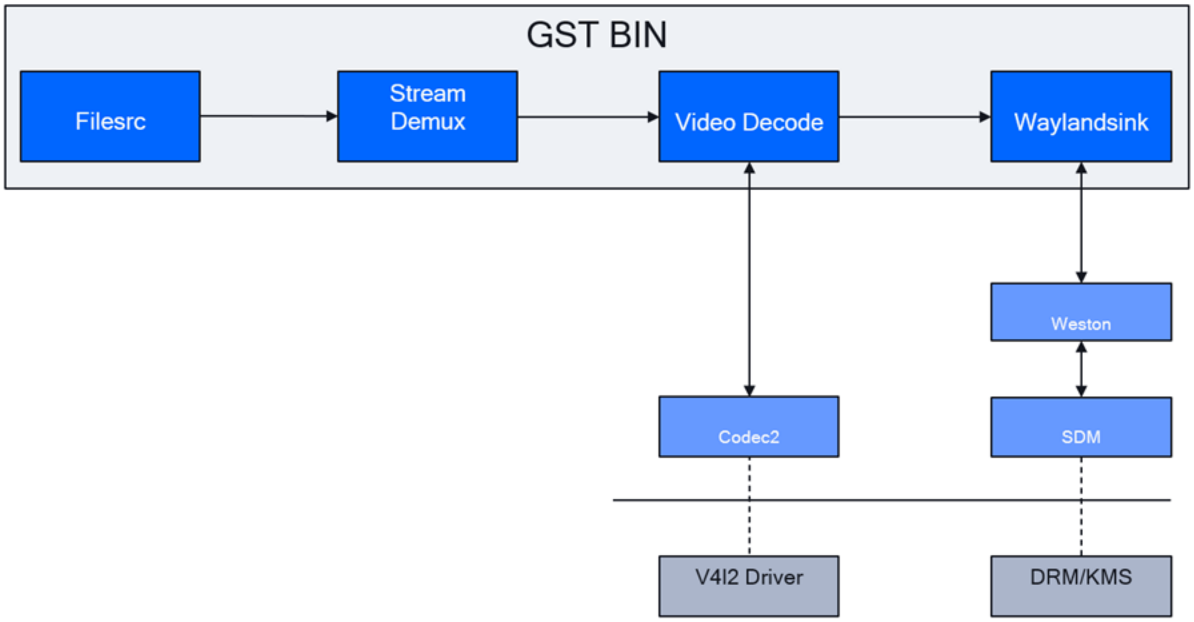
To save the encode bit stream to the file system, connect the encode plug-in to a file muxer (MP4 or MPEGTS).

- Each pad of the camera plug-in corresponds to a separate camera stream. To enable encoding for multiple video streams, multiple instances of the encode plug-ins are connected to different source pads of the camera plug-in
- To enable zero-copy communication, GBM buffers are used between camera plug-in and encode plug-in
- To transfer the encoded streams to a remote device that is connected to the network, the encoded bit streams (output of the video encode plug-in) are connected to the TCP or RTSP sink element
- To provide fine control over the video encode pipeline, the encode parameters are exposed as a property to the application

2.2.2 Video decode

The video decode plug-in, qtic2vdec, connects with the Wayland sink plug-in to implement the playback use case.

Figure 2-2 Video decode architecture



The qtic2vdec plug-in has the following features:

- Uses the Codec2 framework, which internally uses V4L2 input/output controllers (IOCTL) to decode H264/H265 bit streams
- The plug-in is used in playback, transcode, and transform use cases
- For the playback use case, the plug-in connects with the waylandsink plug-in. The element receives the GBM buffers (decoded buffers) as an input, and sends them to the Weston server (via the Wayland protocol) for display composition
- For transcode use cases, the plug-in connects with the video encode plug-in
- For transform (rotate/scale/flip) use cases, the plug-in connects with the transform plug-in, which then connects with the video encode plug-in.
- To provide fine control over the video decode pipeline, the decode parameters are exposed as a property to the application

2.3 Audio

For audio capture and playback, the GStreamer plug-ins (pulsesrc, and pulsesink and pulsedirectsink) use the PulseAudio client to interact with the PulseAudio server.

Audio capture

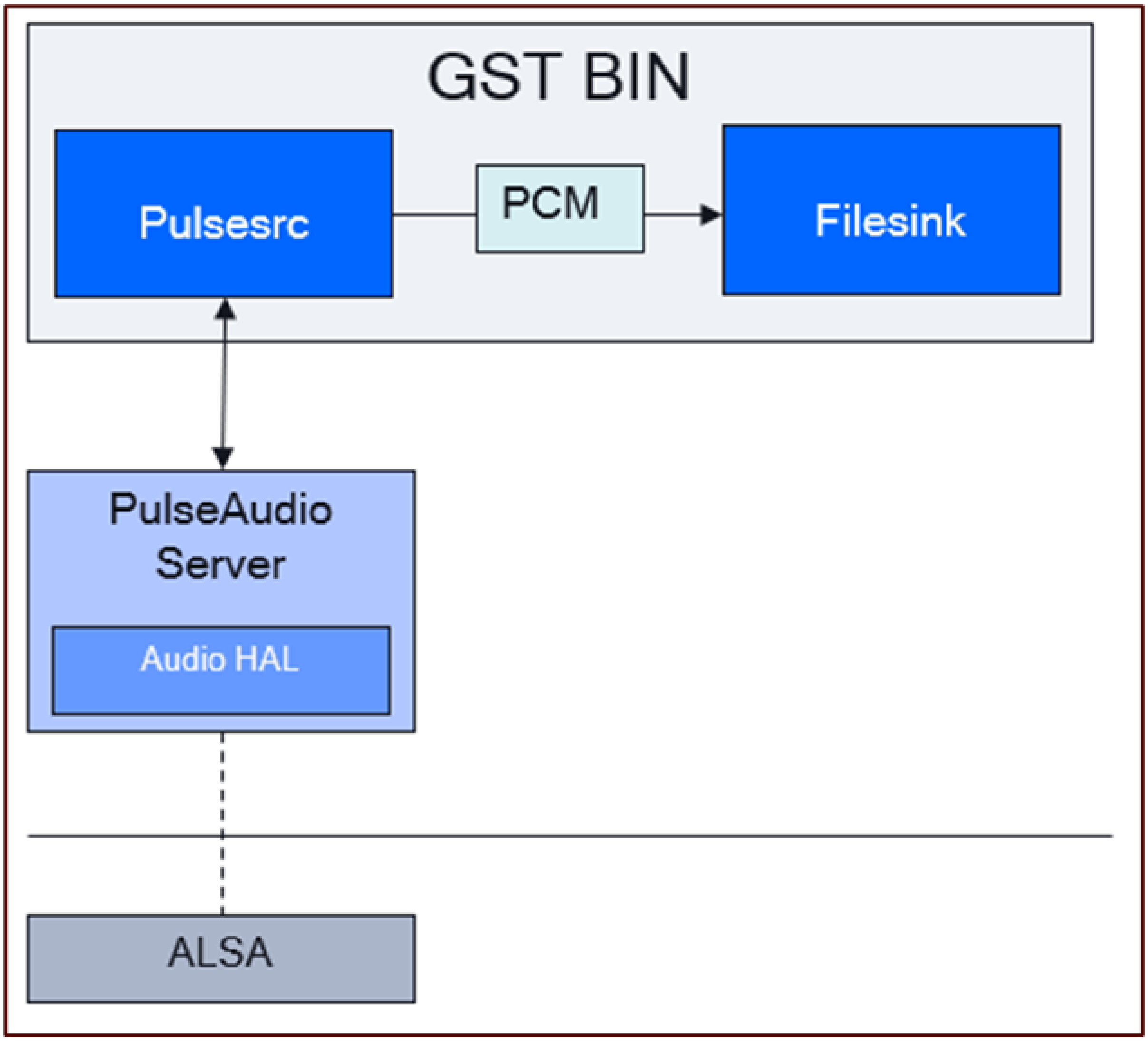


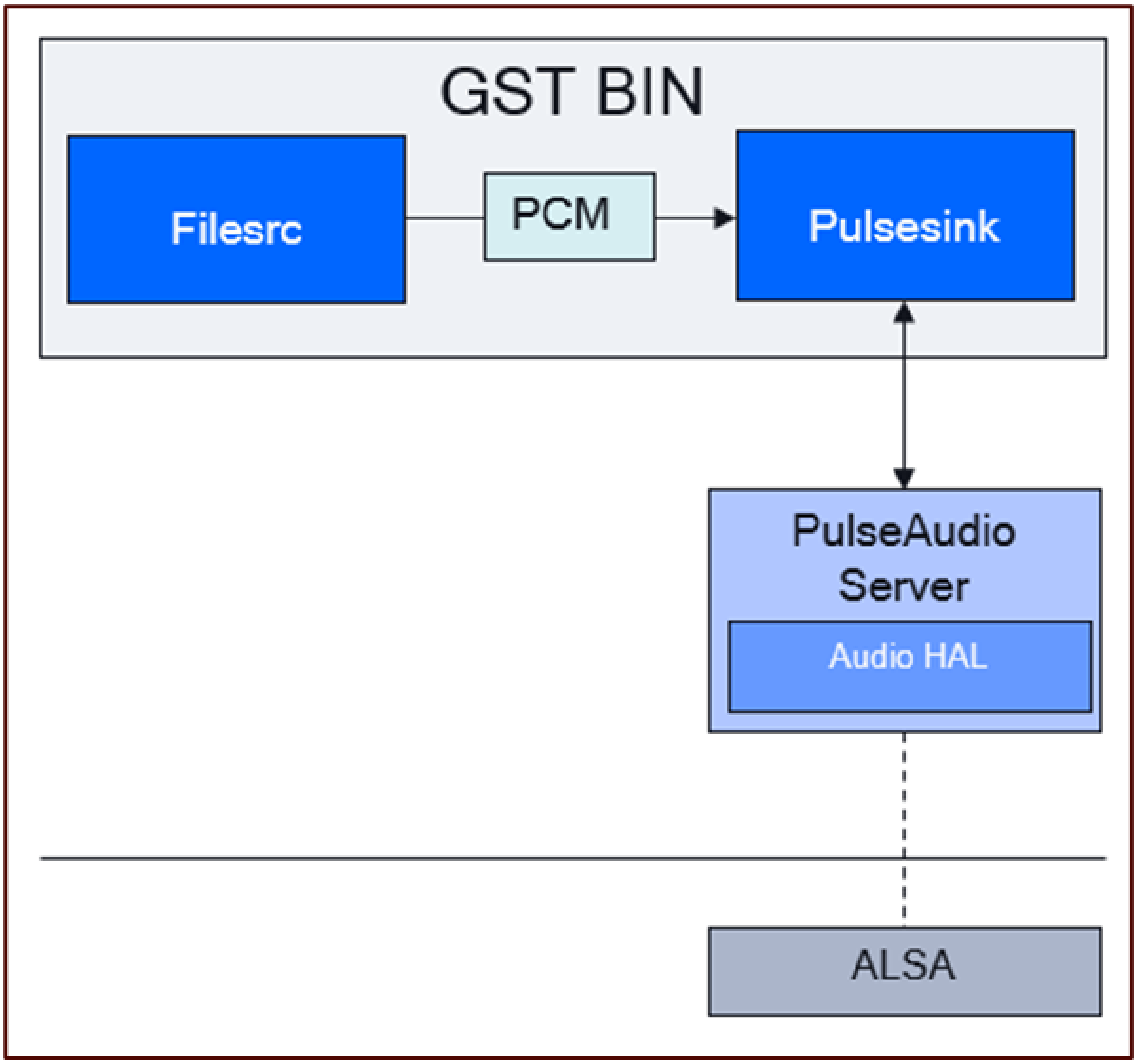
Figure 2-3 Audio capture architecture

- The audio capture plug-in from QIMSDK is offered through the pulsesrc plug-in.
- The pulsesrc plug-in interacts with PulseAudio server to get the PCM audio samples.
- It interacts with the underlying PulseAudio server, which interacts with QTI's hardware through audio HAL.
- PulseAudio server under the hood uses audio HAL as a pluggable module to interact with ALSA driver.

- The user can set the audio device source from which the audio has to be captured
- This plug-in captures raw audio data (PCM) only

Audio playback

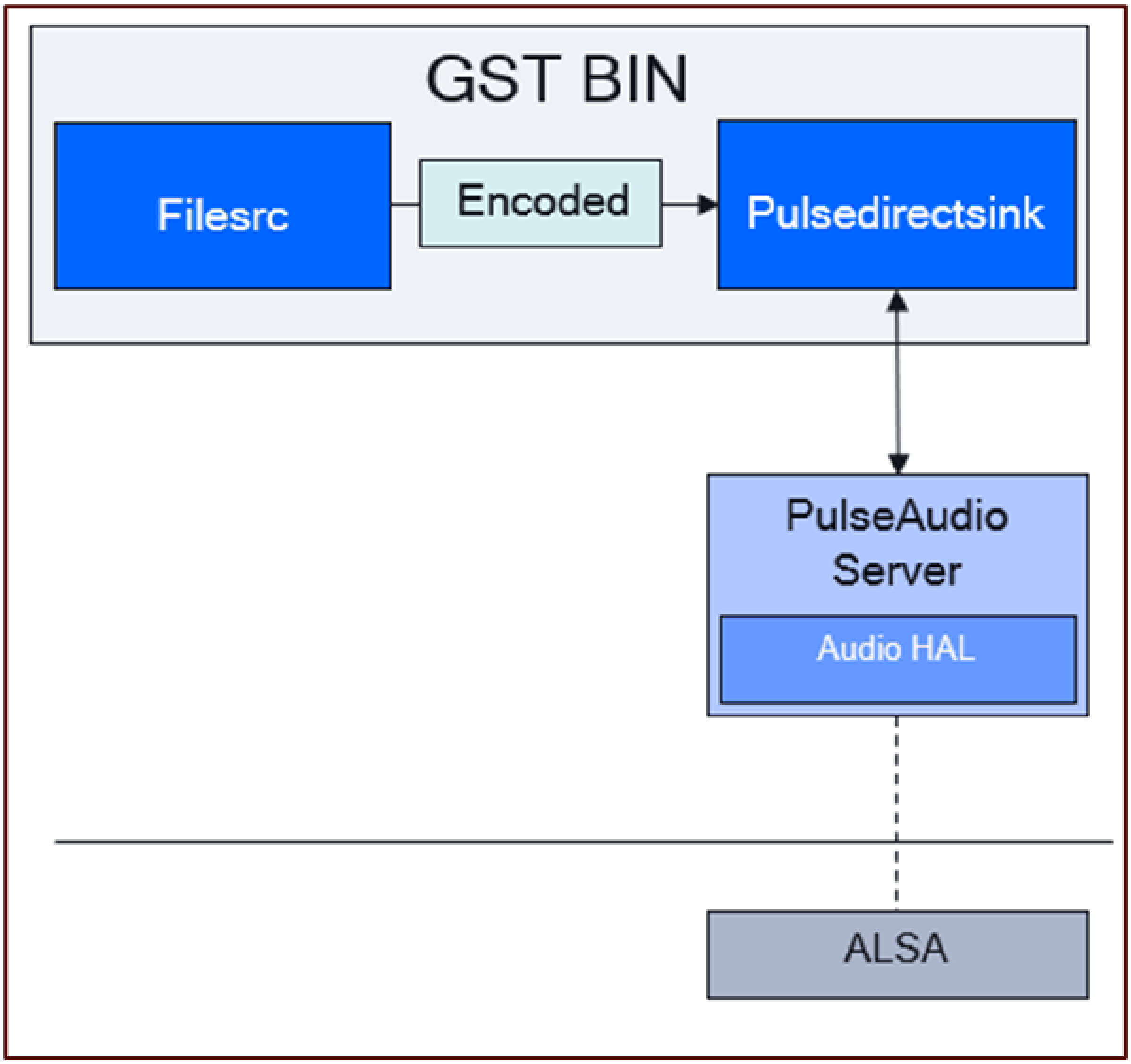
The Audio playback plug-in from QIM-SDK is offered via pulsesink and pulsedirectsink plug-ins.



pulsesink

- pulsesink is an upstream plug-in that enables you to play audio from various audio sources (live source or an encoded audio file)
- It interacts with the underlying PulseAudio server, which interacts with QTI hardware through audio HAL
- Set the audio playback sink to play the audio
- This plug-in plays PCM data only

pulsedirectsink



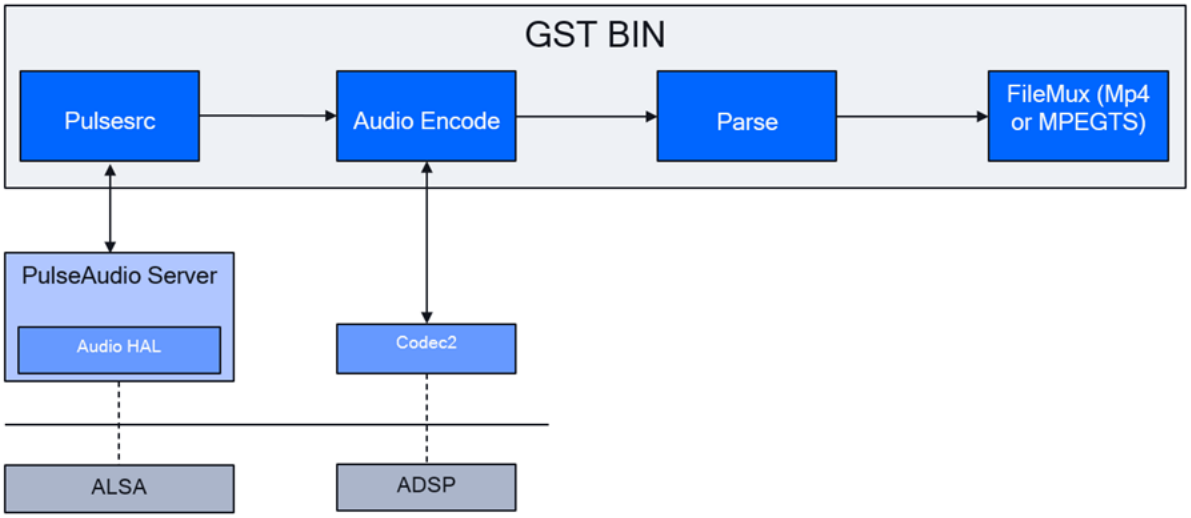
- pulsedirectsink is a plug-in written by QTI. It enables you to play the encoded audio from various audio sources (live source or an encoded audio file)
- It tunnels the encoded data through pulse server, which transmits the data to DSP and manages decoding and playing the stream
- Set the audio playback sink on which the audio has to be played

2.3.1 Audio encode and decode

Audio encode from QIMSDK is offered via qtic2aenc plug-in and decode is offered via qtic2adec plug-in.

Audio encode

Figure 2-4 Audio encode architecture



.

- qtic2aenc is a qualcomm authored plugin which encodes the PCM data captured from a file source or live feed
- It interacts with the underlying Codec2 layer which interacts with ADSP via Audio HAL to encode

NOTE The qtic2aenc plug-in is currently being developed.

Decode

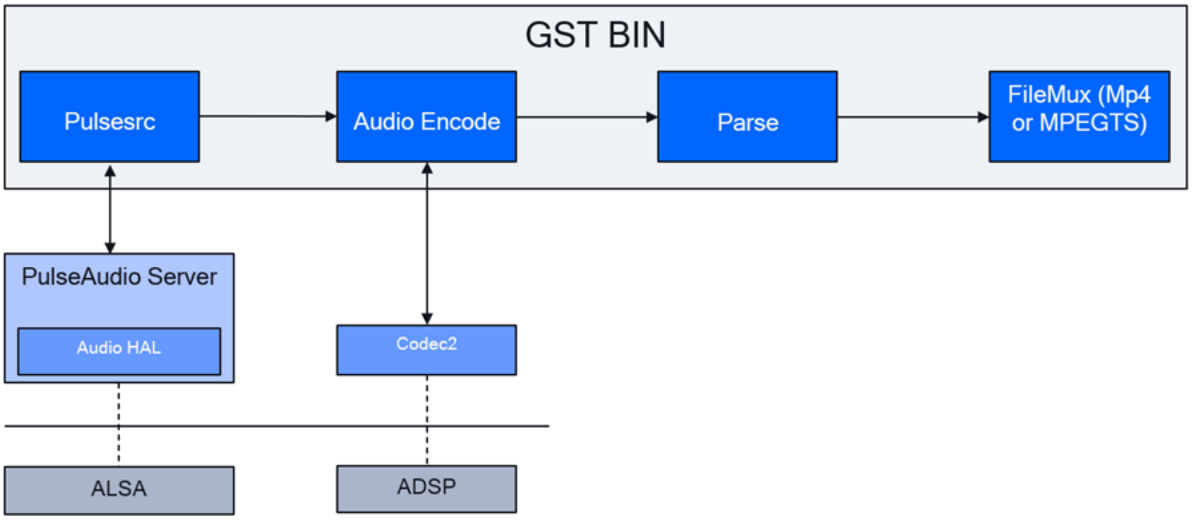


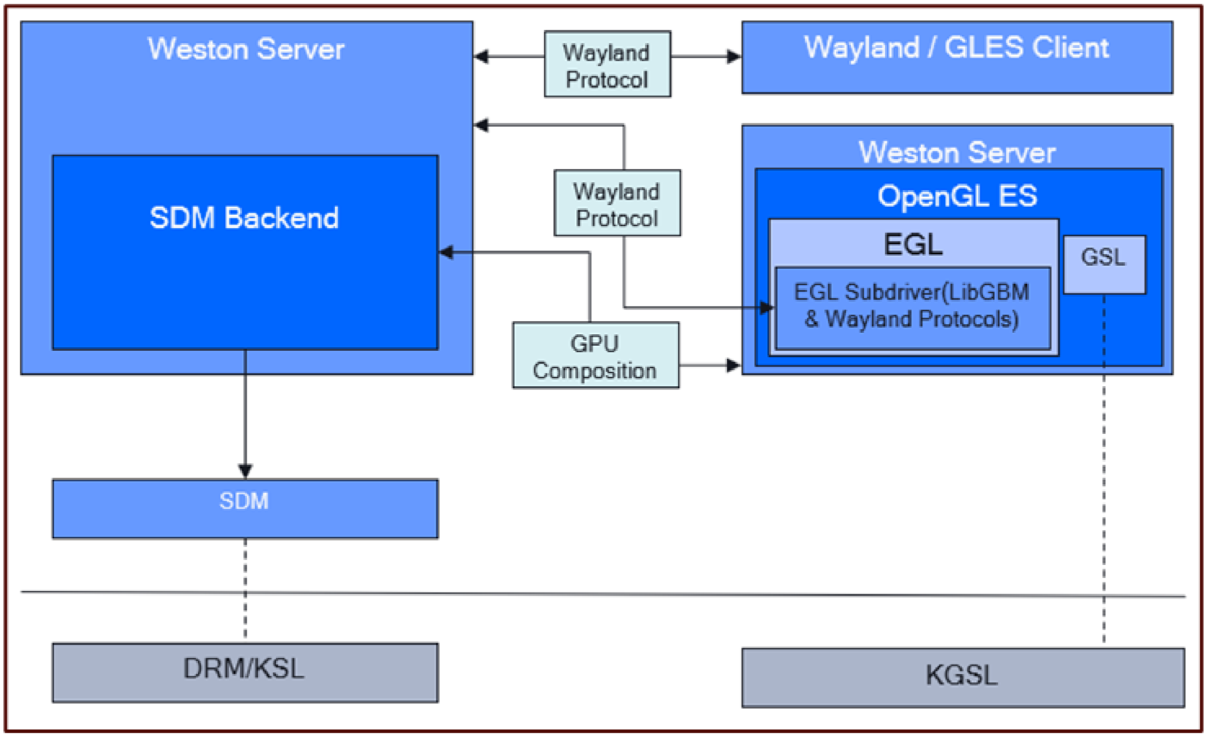
Figure 2-5 Audio decode architecture

- qtic2adec is a qualcomm authored which decodes the data to PCM format which in-turn is for playback/transcode
- It interacts with the underlying Codec2 layer which interacts with ADSP via Audio HAL to decode

NOTE The qtic2adec plug-in is currently being developed.

2.4 Graphics and display

The Weston server supports the graphics and display architecture. Weston is a system-level compositor that uses the Wayland protocol. Weston is dedicated for the management of composition and display with the capability to run as a separate process in the system.



The secure dynamic messaging (SDM) backend of Weston server uses display HAL to interact with display hardware. SDM has multiple platform dependent implementations with one of it designated for direct rendering manager (DRM) and Kernel mode setting (KMS).

The GBM buffer management library (libGBM) includes the ION back end and is used for zero copy buffer sharing between display and graphics. Our EGL platform specific driver (EGL sub-driver) interacts with GBM and Wayland protocols to communicate with Weston compositor.

2.5 Machine learning

QIM SDK 1.0.0 provides support for end-to-end machine learning use case that includes video preprocess, model inference, output tensor postprocess, and inference result overlay to stream.

The machine learning framework provides the following types of video analytics:

image classification

object detection

image segmentation.

With machine learning plug-ins in the framework, you can use TensorFlow Lite and Qualcomm® Neural Processing SDK (formerly known as Snapdragon Neural Processing Engine (SNPE)), and

Cloud AI 100 engine for inferencing. For each of the engines, delegates are enabled to accelerate model inference performance.

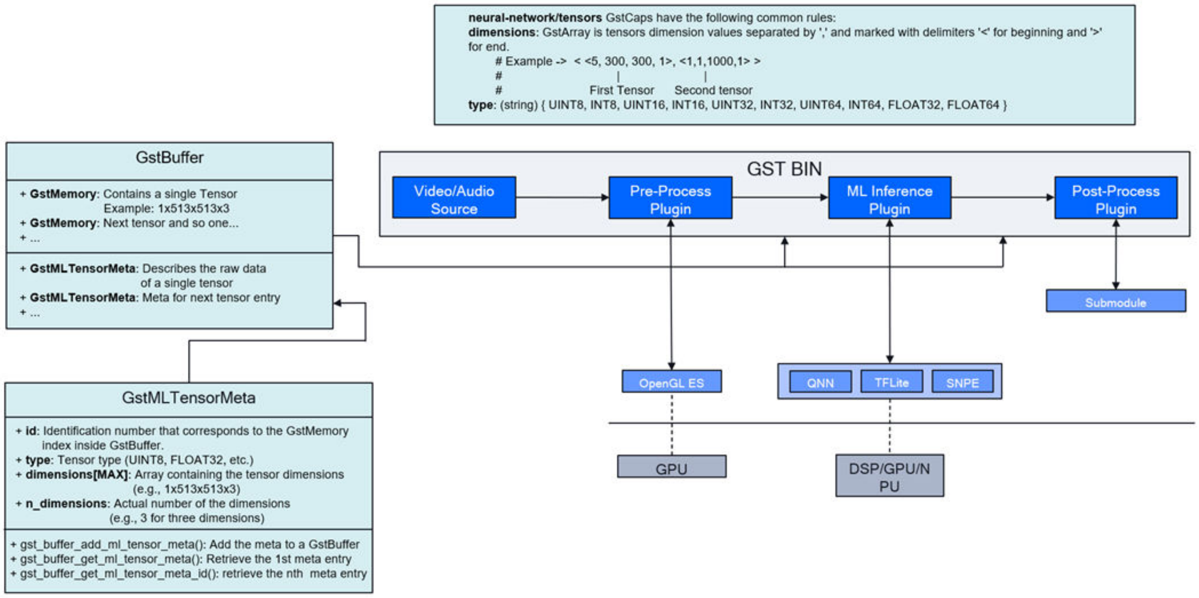

neural-network/tensors GstCaps have the following common rules:
dimensions: GstArray is tensors dimension values separated by ',' and marked with delimiters '<' for beginning and '>' for end.
# Example -> < <5, 300, 300, 1>, <1,1,1000,1> >
# | |
# First Tensor Second tensor
type: (string) { UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64 }
GstBuffer
+ GstMemory: Contains a single Tensor Example: 1x513x513x3
+ GstMemory: Next tensor and so one...
+ ...
+ GstMLTensorMeta: Describes the raw data of a single tensor
+ GstMLTensorMeta: Meta for next tensor entry
+ ...
GstMLTensorMeta
+ id: Identification number that corresponds to the GstMemory index inside GstBuffer.
+ type: Tensor type (UINT8, FLOAT32, etc.)
+ dimensions[MAX]: Array containing the tensor dimensions (e.g., 1x513x513x3)
+ n_dimensions: Actual number of the dimensions (e.g., 3 for three dimensions)
+ gst_buffer_add_ml_tensor_meta(): Add the meta to a GstBuffer
+ gst_buffer_get_ml_tensor_meta(): Retrieve the 1st meta entry
+ gst_buffer_get_ml_tensor_meta_id(): retrieve the nth meta entry
GST BIN
Video/Audio Source
Pre-Process Plugin
ML Inference Plugin
Post-Process Plugin
Submodule
OpenGL ES
QNN
TFLite
SNPE
GPU
DSP/GPU/NPU

Figure 2-6 ML architecture

Machine learning in QIM SDK 1.0.0 is a tensor-based pipeline construction. The source for the pipeline can be YUV streams from live camera source via the qtiqmmfsrc plug-in or any format from an offline video via the filesrc plug-in that will require format conversion. The sink of the pipeline can be on-screen display through waylandsink or filesink saving inferenced video to local storage.

The main section of the machine learning pipeline is the QIM tensor-based machine learning plug-ins. There are three parts to the machine learning section: converter, inference, and postprocess. The qtimlvconverter is responsible for the preprocess of the video stream before sending to the inference plug-in. Preprocess such as color conversion, resize, mean subtraction are all performed in this plug-in. After the conversion is completed, the data is sent to the inference plug-in in a tensor format where the inference plug-in can directly send to machine learning engines for inference.

There are two types of machine learning engines available:

- TensorFlow Lite
- Qualcomm Neural Processing SDK

The qtimltflite is a wrapper that runs the TensorFlow Lite engine, the qtimlsnpe runs the SNPE engine. After inference, the output tensor directly gets passed down to postprocessing plug-ins where each type of machine learning has its own specific plug-in. qtimlvdetection is the postprocess for object detection. qtimlvclassification is the postprocess plug-in for image classification. qtimlvsegmentation is the postprocess plug-in for image segmentation.

The last part of the machine learning pipeline is the overlay plug-in that interprets the machine learning metadata processed in postprocess step. The qtioverlay plug-in draws appropriate overlay on the buffer that allows inference result to be visualized in real-time. The stream can be rendered on to the display, streamed over network, or encoded and stored to local storage.

2.6 Computer vision

The computer vision (CV) module is a hardware block, which is primarily used for image processing. It provides feature likes image preprocessing, object detection, object tracking, motion, and depth processing.

The CV module provides a user space library, which allows the user to use these features. Using QIM SDK 1.0.0, we have enabled the optical flow and pyramid scaler functionality using CV.

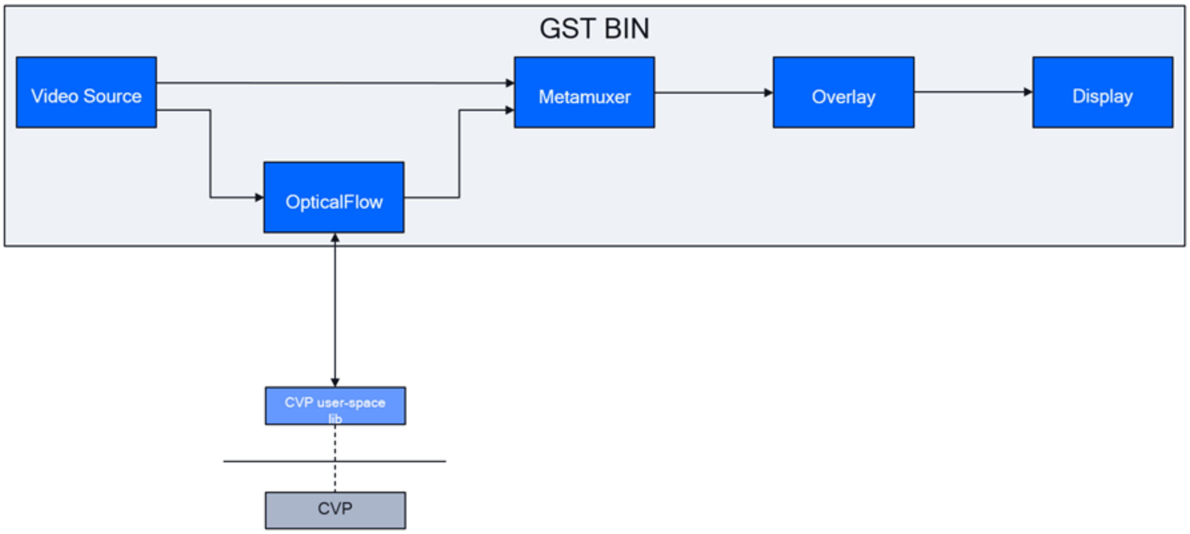


Figure 2-7 CV optical flow pipeline

Optical flow GStreamer plug-in provides motion vector/estimation of moving objects in scene. It leverages CV hardware to calculate motion vector predictions. Input data source can be either live camera or offline stream. The motion vector prediction in form of metadata can be processed by the qtioverlay plug-in to provide arrow visualization on screen or be send to another layer of application for further analysis.

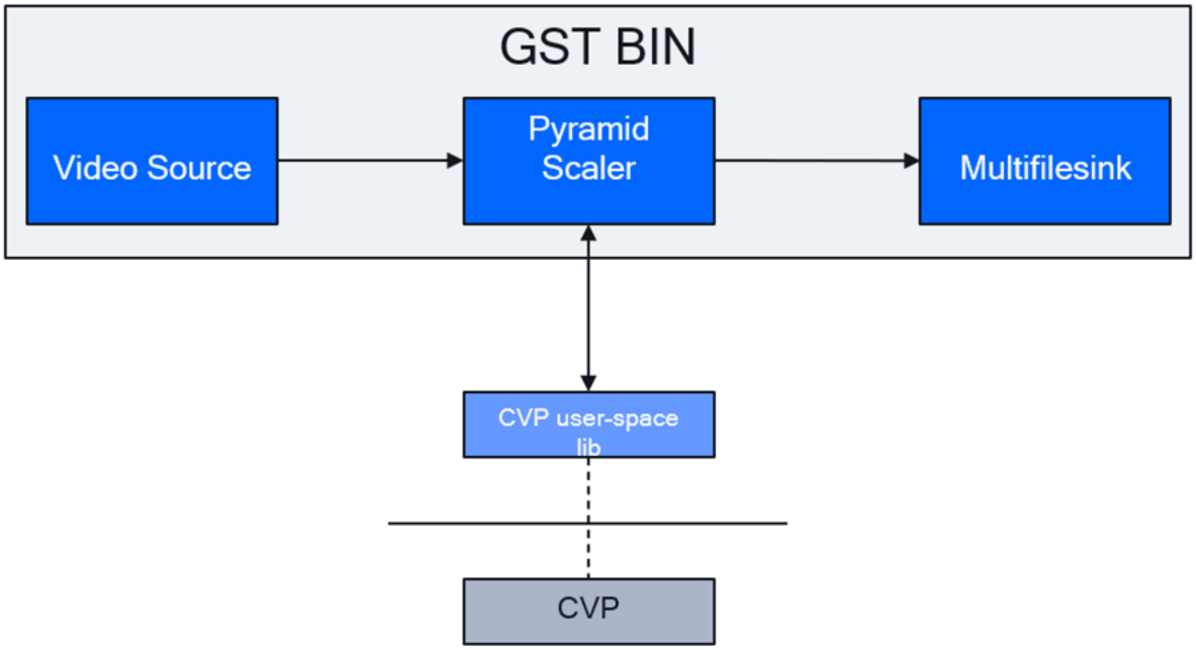


Figure 2-8 CV Image Pyramid Pipeline

Pyramid scaler plug-in provides single or multiple downscaled frames from an input frame based on configuration. It leverages CV hardware to generate the output scaled images. Input data source can be either live camera or offline stream. The output image can be dumped to file, composed, and rendered onto the display, or fed as an input to ML model.

3 QIM SDK plug-ins

The QIM SDK plug-ins are used to control various aspects of multimedia, machine learning, and computer vision.

Plug-in	Usage	Summary	Description
qtiqmmfsrc	**Multimedia**	Camera source plug-in build around the QMMF service to provide multiple 'N' number of streams.	Output streams can be NV12/21, Bayer raw, or NV12/21 UBWC (Universal Bandwidth Compression). The plug-in exposes set of static/ dynamic parameters to control camera IQ parameters and Qualcomm's advanced camera features. It can be connected with different variety of plug-ins based on use case, examples include live camera preview, video encode or live camera machine learning inferencing for both single and multi-streams. This plug-in also supports snapshot, multi-camera, and multi-client use case scenarios. Supported snapshot stream types are NV12/21, Bayer RAW or JPEG blob. Multi-camera use case scenarios are - stereo camera, side-by-side, picture-in-picture, or standalone separate 'N' number of parallel camera streams. Single or multiple different clients can access same or multiple cameras based on the use case. ('N' - depends on capability of specific Qualcomm's chipset and ISP)
pulsesrc	**Multimedia**	Audio (PCM) source plug-in via PulseAudio server	An upstream plug-in which allows to capture PCM samples from any audio source (for example: mic, and so on)
qtivtransform	**Multimedia**	A single input & output plug-in for image/video transformation.	Leverages GPU hardware to perform a number of transformation operations:
qtivcomposer	**Multimedia**	A 'N' inputs to 1 output plug-in for image/video transformation and composition.	Leverages GPU hardware to perform various image composition operations like: Side-by-Side, Picture-in-Picture, alpha blending, etc. In addition to the composition a number of transformation operations can be performed on each 'N' input separately: ■ Downscale/Upscale ■ Rotate (90 CW/CCW, 180) ■ Color convert ■ Flip (vertical/horizontal) ■ Crop

Plug-in	Usage	Summary	Description
qtivsplit	**ML / Multimedia**	A 1 input to 'N' outputs plug-in for image/video stream splitting.	Leverages GPU hardware to duplicate input video stream to multiple outputs or use ROI attached to each incoming buffer to perform cropping and outputting that crop to different stream channel. Crop ROIs or duplicated streams can be downscaled/upscaled with different resolutions and color formats (NV12/21 or RGB). It is widely used in Machine Learning use cases where second level of inferencing needed on specific Region Of Interest (ROI) after first level of inferencing.
qtioverlay	**ML / Multimedia**	Plug-in for drawing different manually set or ML-based overlays over the incoming video frames.	Leverages CPU-based open-source Cairo library for rendering overlays into smaller, more memory efficient buffers and then blend them with the main frame using GPU hardware. The manually set overlays include: ▪ Bounding boxes ▪ Date and/or Time ▪ Buffer timestamp ▪ Custom text ▪ Privacy masks The Machine Learning related overlays are extracted from the metadata attached to each frame. This metadata is attached to the frame by the qtimetamux plug-in. This plug-in will be deprecated soon.
qtimetamux	**ML / CV / Multimedia**	Plug-in for attaching ML string based postprocessing results, CV information, etc. as GstMeta to video/ audio buffers.	Uses frame matching technics to associate/ attach ML string-based postprocessing results (output from postprocessing plug-in) or CV information to original frame as GstMeta. This output can be fed to overlay plug-in for rendering or can be passed down the ML inference pipeline for next level inferencing.
qtisocketsrc	**Multimedia**	Zero copy buffer transfer between clients running in different processes.	Enables zero copy data path between two separate processes running their own GStreamer pipeline. These separate processes/applications can be within native layer or can be within docker execution environment. Socket source plug-in sends the data (buffer file descriptors) through UNIX domain socket.
qtisocketsink	**Multimedia**	Zero copy buffer transfer between clients running in different processes.	Enables zero copy data path between two separate processes running their own GStreamer pipeline. These separate processes/applications can be within native layer or can be within docker execution environment. Socket sink plug-in receives the data (buffer file descriptors) through UNIX domain socket.
qtic2venc	**Multimedia**	H264 Video Encode plug-in that utilizes Codec2	Uses video hardware via Codec2 to get the hardware acceleration to encode input NV12 or NV12 UBWC frames in H264 bit stream. This plug-in can be connected to variety of different plug-ins based on the use case such as MP4 file mux or RTSP, HLS, TCP network streaming.

Plug-in	Usage	Summary	Description
qtic2vdec	**Multimedia**	H264 Video Decode plug-in that utilizes Codec2.	Uses video hardware via Codec2 to get the hardware acceleration to decode input H264 bit stream into NV12 or NV12 UBWC frames. This plug-in can be connected to variety of different plug-ins based on the use case such as MP4 file mux or RTSP, HLS, TCP network streaming.
qtijpegenc	**Multimedia**	JPEG Video Encode plug-in that utilizes Camera Service	The qtijpegenc element utilizes JPEG encoder via camera services to provide hardware accelerated JPEG encoded stream on platforms that support it and is based on GstVideoEncoder base class. This element can work with camera frames (e.g., qtiqmmfsrc) or frames from pre-recorded video file.
pulsesink	**Multimedia**	Audio (PCM) rendering plug-in via PulseAudio server	A standard upstream plug-in which allows to play PCM samples from a PCM file source or from a decoded bit stream.
pulsedirectsink	**Multimedia**	Audio (AAC/MP3) rendering plug-in via PulseAudio server	A plug-in which allows playback of encoded bit stream via tunneling. The encoded data goes as a pass through from PulseAudio server to aDSP where it is decoded and directly sent to playback device (speaker, etc)
waylandsink	**Multimedia**	Display rendering plug-in using weston/wayland server.	A standard wayland sink plug-in, it enables zero copy buffer rendering to display, input buffer can be either from live camera or offline stream. Under the hood it uses standard wayland protocols to communicate with Weston display server. It supports NV12 UBWC which helps in reducing the overall system bus bandwidth.
qticvoptclflow	**CV**	Plug-in for motion estimation using CV OpticalFlow.	Provides motion vector/estimation of moving objects in scene. It leverages CVP hardware to calculate motion vector predictions. Input data source can be either live camera or offline stream. The motion vector prediction in form of metadata can be processed by the qtioverlay plug-in to provide arrow visualization on screen or be send to another layer of application for further analysis.
qticvpimgpyramid	**CV**	Plug-in for generating multiple downscaled frames from an input frame using CVP pyramid downscaler	Provides single or multiple downscaled frames from an input frame based on configuration. It leverages CVP and EVA hardware to generate the output scaled images. Input data source can be either live camera or offline stream. The output image can be dumped to file, composed, and rendered onto the display, or fed as an input to ML model.
qtimlvconverter	**ML**	Plug-in for converting input video/image frames into ML tensors.	Uses GPU hardware acceleration to prepare input frames for ML inferencing by converting them into tensors. Preprocessing operations include: ■ Color convert ■ Downscale/Upscale ■ Aspect Ratio preservation ■ ROI Crop

Plug-in	Usage	Summary	Description
			These tensors flow throughout the pipeline until postprocessing is done by dedicated post-process plug-in.
qtimlsnpe	**ML**	SNPE based inference plug-in.	SNPE based ML inferencing operating with tensors (at both input and output). It receives pre-processed tensors as input and provides inference tensors as output. Output tensors can be provided to dedicated post processing plug-in to be converted into metadata for overlay-use or can be streamed to different source for further analysis based on each use case.
qtimltflite	**ML**	TensorFlow Lite based inference plug-in.	TensorFlow Lite based ML inferencing operating with tensors (at both input and output). It receives pre-processed tensors as input and provides inference tensors as output. Output tensors can be provided to dedicated post processing plug-in to be converted into metadata for overlay-use or can be streamed to different source for further analysis based on each use case.
qtimlvdetection	**ML**	Post-process plug-in for decoding/parsing 'Object Detection' class of inference tensors.	When upstream inference plug-in uses detection modules/models for inferencing, it receives tensor input of detection inference result and parse it into metadata according to the module provided from user. This plug-in works on sub-module methodology where common/generic post processing procedures are part of parent plug-in and ML Model specific post processing is included as part of sub-module. A clean API interface is exposed for application developers to implement their own sub-module for custom postprocessing based on their ML model. SDK provides set of sub-modules as a reference example for open-source ML models.
qtimlvclassification	**ML**	Post-process plug-in for decoding/parsing 'Image Classification' class of inference tensors.	When upstream inference plug-in uses classification modules/models for inferencing, it receives tensor input of detection inference result and parse it into metadata according to the module provided from user. This plug-in works on sub-module methodology where common/generic post processing procedures are part of parent plug-in and ML Model specific post processing is included as part of sub-module. A clean API interface is exposed for application developers to implement their own sub-module for custom postprocessing based on their ML model. SDK provides set of sub-modules as a reference example for open-source ML models.

Plug-in	Usage	Summary	Description
qtimlvsegmentation	**ML**	Post-process plug-in for decoding/parsing 'Image Segmentation' or Depth Estimation class of inference tensors.	It receives tensor input of segmentation inference result and parse it into metadata according to the module provided from user. This plug-in works on sub-module methodology where common/generic post processing procedures are part of parent plug-in and ML Model specific post processing is included as part of sub-module. A clean API interface is exposed for application developers to implement their own sub-module for custom postprocessing based on their ML model. SDK provides set of sub-modules as a reference example for open-source ML models.
qtimlvpose	ML	Post-process plug-in for decoding/parsing 'Pose Estimation' class of inference tensors.	It receives tensor input of segmentation inference result and parse it into metadata according to the module provided from user. This plug-in works on sub-module methodology where common/generic post processing procedures are part of parent plug-in and ML Model specific post processing is included as part of sub-module. A clean API interface is exposed for application developers to implement their own sub-module for custom postprocessing based on their ML model. SDK provides set of sub-modules as a reference example for open-source ML models.

3.1 qtiqmmfsrc

The qtiqmmfsrc element captures video frames through camera service. The plug-in consists of the main class called GstQtiQmmfSrc that acts as a wrapper on top of the QMMF recorder client with separate pads for video and image streams.

The pads store the creation time parameters (passed as GstCaps during pipeline creation) for the particular stream while the GstQtiQmmfSrc takes that information, translates it to the QMMF Recorder Client parameters and calls the necessary APIs on each state transition of the element. For video and image pads, the camera device ID, which will be used for this instance of the plug-in can be set via the “camera” property (by default this ID is 0).

- During transition between NULL and READY state, the plug-in opens and initializes the camera device with the given ID.
- When transitioning to PAUSED state from READY, the plug-in translates the set pad parameters and makes calls to the QMMF service in order to create the source streams for each pad.
- The session streams are started when transition is done to PLAYING state.

When a frame is received in the main class, it will create a GstBuffer and send it to the relevant pad buffer queue. The pad will push the buffer to its linked sink pad from the next plug-in. Buffer allocation takes place inside the QMMF service while the plug-in only ensures that the buffers are returned to the service when they are no longer in use.

For video, buffers are sent by the QMMF service when the plug-in state is PLAYING, but for image pad a "capture-image" signal must be sent. For each "capture-image" a single buffer will be sent from the QMMF service.

Inheritance chain: GObject → GstObject → GstElement → GstQmmfSrc

qtiqmmfsrc pad configuration

Table 3-1 Pad templates for qtiqmmfsrc

Pad Name	Capabilities		
SRC template: 'image_ %u' *Availability:* On request *Direction:* source	image/jpeg	width: height: framerate:	[16, 9248] [16, 6944] [0/1, 30/1]
	video/x-raw	format: width: height: framerate:	{ (string)NV12, (string)NV21 } [16, 9248] [16, 6944] [0/1, 30/1]
	video/x-raw(memory:GBM)	format: width: height: framerate:	{ (string)NV12, (string)NV21 } [16, 9248] [16, 6944] [0/1, 30/1]
	video/x-bayer	format: bpp: width: height: framerate:	{ (string)bggr, (string)rggb, (string)gbrg, (string)grbg, (string)mono } { (string)8, (string)10, (string)12, (string)16 } [16, 9248] [16, 6944] [0/1, 30/1]
SRC template: 'video_ %u' *Availability:* On request *Direction:* source	image/jpeg	width: height: framerate:	[16, 9248] [16, 6944] [0/1, 480/1]
	video/x-raw	format: width: height: framerate:	{ (string)NV12, (string)NV16, (string)UYVY, (string)P010_10LE, (string)NV12_10LE32 } [16, 9248] [16, 6944] [0/1, 480/1]
	video/x-raw(memory:GBM)	format: width: height: framerate:	(string)NV12, (string)NV16, (string)UYVY, (string)P010_10LE, (string)NV12_10LE32 } [16, 9248] [16, 6944] [0/1, 480/1]
	video/x-bayer	format: bpp: width: height: framerate:	{ (string)bggr, (string)rggb, (string)gbrg, (string)grbg, (string)mono } { (string)8, (string)10, (string)12, (string)16 } [16, 9248] [16, 6944] [0/1, 480/1]

Table 3-2 Pad properties of qtiqmmfsrc

Pad	Property	Description
'video_%u'	source-index	▪ Index of the source video pad to which this pad will be linked ▪ flags: readable, writable, changeable only in NULL or READY state ▪ Integer. Range: -1 - 2147483647 Default: -1
	framerate	▪ Target framerate in frames per second for displaying ▪ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ▪ Double. Range: 0 - 30 Default: 30
	crop	▪ Crop rectangle ('<X, Y, WIDTH, HEIGHT>'). Applicable only for JPEG and YUY2 formats ▪ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ▪ GstValueArray of GValues of type "gint"
	extra-buffers	▪ Number of additional buffers that will be allocated. ▪ flags: readable, writable, changeable only in NULL or READY state ▪ Unsigned Integer. Range: 0 - 4294967295 Default: 0

qtiqmmfsrc element configuration

Table 3-3 Element properties for qtiqmmfsrc

Property	Description
name	▪ The name of the object ▪ flags: readable, writable ▪ String. Default: "qmmfsrc0"
parent	▪ The parent of the object ▪ flags: readable, writable ▪ Object of type "GstObject"
camera	▪ Camera device ID to be used by video/image pads ▪ flags: readable, writable ▪ Unsigned Integer. Range: 0 - 10 Default: 0
slave	▪ Set camera as slave device ▪ flags: readable, writable ▪ Boolean. Default: false
ldc	▪ Lens Distortion Correction ▪ flags: readable, writable ▪ Boolean. Default: false
lcac	▪ Lateral Chromatic Aberration Correction ▪ flags: readable, writable ▪ Boolean. Default: false
eis	▪ Electronic Image Stabilization to reduce the effects of camera shake ▪ flags: readable, writable ▪ Boolean. Default: false

Table 3-3 Element properties for qtiqmmfsrc (cont.)

Property	Description
shdr	■ Super High Dynamic Range Imaging ■ flags: readable, writable ■ Boolean. Default: false
adrc	■ Automatic Dynamic Range Compression ■ flags: readable, writable ■ Boolean. Default: false
control-mode	■ Overall mode of 3A (auto-exposure, auto-white-balance, auto-focus) control routines. This is a top-level 3A control switch. When set to OFF, all 3A control by the camera device is disabled. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraControlMode" Default: 1, "auto" ■ (0): off - Full application control of pipeline. ■ (1): auto - Manual control of capture parameters is disabled. ■ (2): use-scene-mode - Use a specific scene mode. ■ (3): off-keep-state - Same as OFF mode, except that this capture will not be used by camera device background auto-exposure, auto-white balance and auto-focus algorithms (3A) to update their statistics.
effect	■ Effect applied on the camera frames ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraEffectMode" Default: 0, "off" ■ (0): off - No color effect will be applied. ■ (1): mono - A 'monocolor' effect where the image is mapped into a single color. ■ (2): negative - A 'photo-negative' effect where the image's colors are inverted. ■ (3): solarize - A 'solarisation' effect (Sabattier effect) where the image is wholly or partially reversed in tone. ■ (4): sepia - A 'sepia' effect where the image is mapped into warm gray, red, and brown tones. ■ (5): posterize - A 'posterization' effect where the image uses discrete regions of tone rather than a continuous gradient of tones. ■ (6): whiteboard - A 'whiteboard' effect where the image is typically displayed as regions of white, with black or grey details. ■ (7): blackboard - A 'blackboard' effect where the image is typically displayed as regions of black, with white or grey details. ■ (8): aqua - An 'aqua' effect where a blue hue is added to the image.
scene	■ Camera optimizations depending on the scene ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraSceneMode" Default: 1, "face-priority" ■ (0): disabled - Indicates that no scene modes are set. ■ (1): face-priority - Optimized for photos of with priority of people faces. ■ (2): action - Optimized for photos of quickly moving objects. ■ (3): portrait - Optimized for still photos of people. ■ (4): landscape - Optimized for photos of distant macroscopic objects. ■ (5): night - Optimized for low-light settings. ■ (6): night-portrait - Optimized for still photos of people in low-light settings.

Table 3-3 Element properties for qtiqmmfsrc (cont.)

Property	Description
	■ (7): theatre - Optimized for dim, indoor settings where flash must remain off. ■ (8): beach - Optimized for bright, outdoor beach settings. ■ (9): snow - Optimized for bright, outdoor settings containing snow. ■ (10): sunset - Optimized for scenes of the setting sun. ■ (11): steady-photo - Optimized to avoid blurry photos due to small amounts of device motion (for example: due to hand shake). ■ (12): fireworks - Optimized for nighttime photos of fireworks. ■ (13): sports - Optimized for photos of quickly moving people. ■ (14): party - Optimized for dim, indoor settings with multiple moving people. ■ (15): candlelight - Optimized for dim settings where the main light source is a candle. ■ (16): hdr - Turn on a device-specific high dynamic range (HDR) mode.
antibanding	■ Camera antibanding routine for the current illumination condition ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstAntibandingMode" Default: 3, "auto" ■ (0): off - The camera device will not adjust exposure duration to avoid banding problems. ■ (1): 50 Hz - The camera device will adjust exposure duration to avoid banding problems with 50 Hz illumination sources. ■ (2): 60 Hz - The camera device will adjust exposure duration to avoid banding problems with 60 Hz illumination sources. ■ (3): auto - The camera device will automatically adapt its antibanding routine to the current illumination condition.
sharpness	■ Image Sharpness Strength ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Integer. Range: 0 - 6 Default: 2
contrast	■ Image Contrast Strength ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Integer. Range: 1 - 10 Default: 5
saturation	■ Image Saturation Strength ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Integer. Range: 0 - 10 Default: 5
iso-mode	ISO exposure mode ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraISOMode" Default: 0, "auto" ■ (0): auto - The ISO exposure mode will be chosen depending on the scene. ■ (1): deblur - The ISO exposure sensitivity set to prioritize motion deblur. ■ (2): 100 - The ISO exposure sensitivity set to prioritize level 100. ■ (3): 200 - The ISO exposure sensitivity set to prioritize level 200. ■ (4): 400 - The ISO exposure sensitivity set to prioritize level 400. ■ (5): 800 - The ISO exposure sensitivity set to prioritize level 800. ■ (6): 1600 - The ISO exposure sensitivity set to prioritize level 1600. ■ (7): 3200 - The ISO exposure sensitivity set to prioritize level 3200. ■ (8): manual - The ISO exposure value provided by manual-iso-value will be used.

Table 3-3 Element properties for qtiqmmfsrc (cont.)

Property	Description
manual-iso-value	■ Manual exposure ISO value. Used when the ISO mode is set to 'manual' ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Integer. Range: 100 - 3200 Default: 800
exposure-mode	■ The desired mode for the camera's exposure routine. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraExposureMode" Default: 1, "auto" ■ (0): off - The auto exposure routine is disabled. Manual exposure time will be used set via the 'exposure-time' property ■ (1): auto - The auto exposure routine is active.
exposure-lock	■ Locks current camera exposure routine values from changing. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Boolean. Default: false
exposure-metering	■ The desired mode for the camera's exposure metering routine. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraExposureMetering" Default: 0, "average" ■ (0): average - The camera device's exposure metering is calculated as average from the whole frame. ■ (1): center-weighted - The camera device's exposure metering is calculated from the center region of the frame. ■ (2): spot - The camera device's exposure metering is calculated from a chosen spot. ■ (6): custom - The camera device's exposure metering is calculated from a custom metering table.
exposure-compensation	■ Adjust (Compensate) camera images target brightness. Adjustment is measured as a count of steps. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Integer. Range: -12 - 12 Default: 0
manual-exposure-time	■ Manual exposure time in nanoseconds. Used when the Exposure mode is set to 'off'. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Integer64. Range: 0 - 9223372036854775807 Default: 33333333
custom-exposure-table	■ A GstStructure describing custom exposure table ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: "org.codeaurora.qcamera3.exposuretable;"
white-balance-mode	The desired mode for the camera's white balance routine. flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraWiteBalanceMode" Default: 3, "auto" ■ (0): off - Both auto and manual white balance routines are disabled. ■ (1): manual-cc-temp - The auto-white balance routine is inactive and manual color correction temperature is used which is set via the 'manual-wb-settings' property. ■ (2): manual-rgb-gains - The auto-white balance routine is inactive and manual R/G/B gains are used which are set via the 'manual-wb-settings' property. ■ (3): auto - The auto-white balance routine is active. ■ (4): shade - The camera device uses shade light as the assumed scene illumination for white balance correction.

Table 3-3 Element properties for qtiqmmfsrc (cont.)

Property	Description
	■ (5): incandescent - The camera device uses incandescent light as the assumed scene illumination for white balance correction. ■ (6): fluorescent - The camera device uses fluorescent light as the assumed scene illumination for white balance correction. ■ (7): warm-fluorescent - The camera device uses warm fluorescent light as the assumed scene illumination for white balance correction. ■ (8): daylight - The camera device uses daylight light as the assumed scene illumination for white balance correction. ■ (9): cloudy-daylight - The camera device uses cloudy daylight light as the assumed scene illumination for white balance correction. ■ (10): twilight - The camera device uses twilight light as the assumed scene illumination for white balance correction.
white-balance-lock	■ Locks current White Balance values from changing. Affects only non-manual white balance modes. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Boolean. Default: false
manual-wb-settings	■ Manual White Balance settings such as color correction temperature and R/G/B gains. Used in manual white balance modes. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: "org.codeaurora.qcamera3.manualWB;"
focus-mode	■ Whether auto-focus is currently enabled, and in what mode it is. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraFocusMode" Default: 0, "off" ■ (0): off - The auto focus routine is disabled. ■ (1): auto - The auto focus routine is active. ■ (2): macro - In this mode, the auto focus algorithm is optimized for focusing on objects very close to the camera. ■ (3): continuous - In this mode, the AF algorithm modifies the lens position continually to attempt to provide a constantly-in-focus image stream. ■ (4): edof - The camera device will produce images with an extended depth of field automatically; no special focusing operations need to be done before taking a picture.
noise-reduction	■ Noise reduction filter mode ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraNoiseReduction" Default: 1, "fast" ■ (0): off - No noise reduction filter is applied. ■ (1): fast - TNR (Temporal Noise Reduction) Fast Mode. ■ (2): hq - TNR (Temporal Noise Reduction) High Quality Mode.
noise-reduction-tuning	■ A GstStructure describing noise reduction tuning ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: "org.quic.camera.anr_tuning;"
zoom	■ Camera zoom rectangle ('<X, Y, WIDTH, HEIGHT >') in sensor active pixel array coordinates ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ GstValueArray of GValues of type "gint"

Table 3-3 Element properties for qtiqmmfsrc (cont.)

Property	Description
defog-table	■ A GstStructure describing defog table ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: "org.quic.camera.defog;"
ltm-data	■ A GstStructure describing local tone mapping data ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: "org.quic.camera.ltmDynamicContrast;"
infrared-mode	■ Infrared Mode ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstCameraIRMode" Default: 0, "off" ■ (0): off - The infrared LED is OFF and cut filter is applied i.e. infrared light is blocked. ■ (1): on - The infrared LED is ON and cut filter is removed i.e. infrared light is allowed. ■ (2): auto - The infrared LED and cut filter are turned ON or OFF depending on the conditions. ■ (3): cut-filter-only - The infrared LED is turned OFF and cut filter is applied i.e. IR light is blocked. ■ (4): cut-filter-disable - Infrared cut filter is removed allowing IR light to pass. This mode is used for transitioning from 'cut-filter-only' mode i.e., disabling only the cut filter.
active-sensor-size	■ The active pixel array of the camera sensor ('<X, Y, WIDTH, HEIGHT >') and it is filled only when the plug-in is in READY or above state ■ flags: readable, changeable in NULL, READY, PAUSED or PLAYING state ■ GstValueArray of GValues of type "gint"
sensor-mode	■ Force set Sensor Mode index (0-15). -1 for Auto selection ■ flags: readable, writable ■ Integer. Range: -1 - 15 Default: -1
static-metadata	■ Supported camera capabilities as Android CameraMetadata object. Caller is responsible for releasing the object. ■ flags: readable, changeable in NULL, READY, PAUSED or PLAYING state ■ Pointer.
video-metadata	■ Settings and parameters used for submitting capture requests for video streams in the form of Android CameraMetadata object. Caller is responsible for releasing the object. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Pointer.
image-metadata	■ Settings and parameters used for submitting capture requests for high quality images via the capture-image signal in the form of Android CameraMetadata object. Caller is responsible for releasing the object. ■ flags: readable, changeable in NULL, READY, PAUSED or PLAYING state ■ Pointer.
exposure-compensation	■ Adjust (Compensate) camera images target brightness. Adjustment is measured as a count of steps. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Integer. Range: -12 - 12 Default: 0

Table 3-3 Element properties for qtiqmmfsrc (cont.)

Property	Description
exposure-compensation-for-each	■ Set camera capture images exposure for each capture image. Format such as:<0,0,-2,2,-4,4,-6,6,10,-10>. ARRAY LENGTH SHOULD BE 10!!! ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ GstValueArray of GValues of type "gint"
frc-mode	■ Stream frame rate control mode. ■ flags: readable, writable Enum "GstFrcMode" Default: 0, "frame-skip" ■ (0): frame-skip - Control stream frame rate by frame skip ■ (1): capture-request - Control stream frame rate by camera capture request
ife-direct-stream	■ IFE direct stream support, with this param, ISP will generate output stream from IFE directly and skip others ISP modules like IPE ■ flags: readable, writable ■ Boolean. Default: false
op-mode	■ Camera Operation mode, with this param, camera will work on specified mode ■ Flags: readable, writable Enum “GstCamOpMode” Default:0, “none” ■ (0): none – no camera mode specified ■ (1): frame selection – pick frame in camera pipeline ■ (2): fastswitch – switch sensor mode between preview and preview plus video

Table 3-4 Element signals for qtiqmmfsrc

Signal	Function
result-metadata	void user_function (GstElement* object, gpointer arg0, gpointer user_data);
urgent-metadata	void user_function (GstElement* object, gpointer arg0, gpointer user_data);

Table 3-5 Element actions for qtiqmmfsrc

Action	Function
capture-image	gboolean user_function (GstElement* object, GstImageCaptureMode arg0, guint arg1, GPtrArray* arg2);
cancel-capture	gboolean user_function (GstElement* object);

Usage

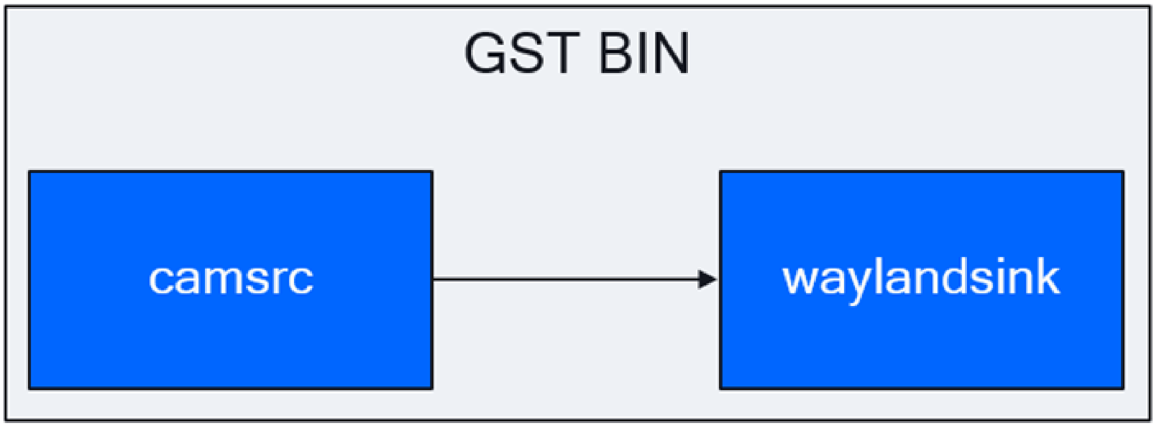


Figure 3-1 Live preview

Command to connect to display:

```
export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 --gst-debug=2
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink
fullscreen=true async=true sync=false
```

3.2 pulsesrc

The pulsesrc GStreamer plug-in interacts with the underlying PulseAudio sound server to capture the pulse coded modulation (PCM) samples.

The PulseAudio server interacts with the underlying hardware to provide the capture capability and is based on GstAudioSrc base class.

Inheritance chain: GObject → GstObject → GstElement → GstBaseSrchttps://gstreamer.freedesktop.org/documentation/base/gstbasesink.html?gi-language=c#GstBaseSink→ GstPushSrc → GstAudioBaseSrc → GstAudioSrc → GstPulseSrc

pulsesrc pad configuration

Table 3-6 Pad templates for pulsesrc

Pad Name	Capabilities		
SRC template: 'src' *Availability:* Always *Direction:* src	audio/x-raw	format:	{ (string)S16LE, (string)S16BE, (string)F32LE, (string)F32BE, (string)S32LE, (string)S32BE, (string)S24LE, (string)S24BE, (string)S24_32LE, (string)S24_32BE, (string)U8 }
		layout:	interleaved
		rate:	[1, 22579200]
		channels:	[1, 32]
	audio/x-alaw	rate:	[1, 22579200]
		channels:	[1, 32]

Table 3-6 Pad templates for pulsesrc (cont.)

Pad Name	Capabilities		
	audio/x-mulaw	rate:	[1, 22579200]
		channels:	[1, 32]

pulsesrc element configuration

Table 3-7 Element properties of pulsesrc

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "pulsesrc0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
blocksize	■ Size in bytes to read per buffer (-1 = default) ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
num-buffers	■ Number of buffers to output before sending EOS (-1 = unlimited) ■ flags: readable, writable ■ Integer. Range: -1 - 2147483647 Default: -1
typefind	■ Run type find before negotiating (deprecated, non-functional) ■ flags: readable, writable, deprecated ■ Boolean. Default: false
do-timestamp	■ Apply current stream time to buffers ■ flags: readable, writable ■ Boolean. Default: true
buffer-time	■ Size of audio buffer in microseconds. This is the maximum amount of data that is buffered in the device and the maximum latency that the source reports. This value might be ignored by the element if necessary; see "actual-buffer-time" ■ flags: readable, writable ■ Integer64. Range: 1 - 9223372036854775807 Default: 200000
latency-time	■ The minimum amount of data to read in each iteration in microseconds. This is the minimum latency that the source reports. This value might be ignored by the element if necessary; see "actual-latency-time" ■ flags: readable, writable ■ Integer64. Range: 1 - 9223372036854775807 Default: 10000
actual-buffer-time	■ Actual configured size of audio buffer in microseconds ■ flags: readable ■ Integer64. Range: -1 - 9223372036854775807 Default: -1
actual-latency-time	■ Actual configured audio latency in microseconds ■ flags: readable ■ Integer64. Range: -1 - 9223372036854775807 Default: -1

Table 3-7 Element properties of pulsesrc (cont.)

Property	Description
provide-clock	■ Provide a clock to be used as the global pipeline clock ■ flags: readable, writable ■ Boolean. Default: true
slave-method	■ Algorithm used to match the rate of the master clock ■ flags: readable, writable ■ Enum "GstAudioBaseSrcSlaveMethod" Default: 2, "skew" □ (0): resample - GST_AUDIO_BASE_SRC_SLAVE_RESAMPLE □ (1): re-timestamp - GST_AUDIO_BASE_SRC_SLAVE_RE_TIMESTAMP □ (2): skew - GST_AUDIO_BASE_SRC_SLAVE_SKEW □ (3): none - GST_AUDIO_BASE_SRC_SLAVE_NONE
server	■ The PulseAudio server to connect to ■ flags: readable, writable ■ String. Default: null
device	■ The PulseAudio source device to connect to ■ flags: readable, writable ■ String. Default: null
current-device	■ The current PulseAudio source device ■ flags: readable ■ String. Default: ""
device-name	■ Human-readable name of the sound device ■ flags: readable ■ String. Default: null
volume	■ Linear volume of this stream, 1.0=100% ■ flags: readable, writable ■ Double. Range: 0 - 10 Default: 1
mute	■ Mute state of this stream ■ flags: readable, writable ■ Boolean. Default: false
client-name	■ The PulseAudio client_name_to_use ■ flags: readable, writable, changeable only in NULL or READY state ■ String. Default: "gst-inspect-1.0"
stream-properties	■ list of PulseAudio stream properties ■ flags: readable, writable ■ Boxed pointer of type "GstStructure"

Table 3-7 Element properties of pulsesrc (cont.)

Property	Description
source-output-index	■ The index of the PulseAudio source output corresponding to this record stream ■ flags: readable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4294967295
stream-flags	■ Stream flags to use (not all default flags can be overridden) ■ flags: writable ■ Flags "GstPulseStreamFlags" Default: 0x0000000f, "Timing update requests are issued periodically automatically+Don't force the time to increase monotonically+Interpolate the latency for this stream+Create the stream corked" □ ((0x00000000): Flag to pass when no specific options are needed - GST_PULSE_STREAM_NOFLAGS □ (0x00000001): Create the stream corked - GST_PULSE_STREAM_START_CORKED □ (0x00000002): Interpolate the latency for this stream - GST_PULSE_STREAM_INTERPOLATE_TIMING □ (0x00000004): Don't force the time to increase monotonically - GST_PULSE_STREAM_NOT_MONOTONIC □ (0x00000008): Timing update requests are issued periodically automatically - GST_PULSE_STREAM_AUTO_TIMING_UPDATE □ (0x00000010): Don't remap channels by their name, instead map them simply by their index - GST_PULSE_STREAM_NO_REMAP_CHANNELS □ (0x00000020): When remapping channels by name, don't upmix or downmix them to related channels - GST_PULSE_STREAM_NO_REMIX_CHANNELS □ (0x00000040): Use the sample format of the sink/device this stream is being connected to - GST_PULSE_STREAM_FIX_FORMAT □ (0x00000080): Use the sample rate of the sink - GST_PULSE_STREAM_FIX_RATE □ (0x00000100): Use the number of channels and the channel map of the sink - GST_PULSE_STREAM_FIX_CHANNELS □ (0x00000200): Don't allow moving of this stream to another sink/device - GST_PULSE_STREAM_DONT_MOVE □ (0x00000400): Allow dynamic changing of the sampling rate during playback - GST_PULSE_STREAM_VARIABLE_RATE □ (0x00000800): Find peaks instead of resampling - GST_PULSE_STREAM_PEAK_DETECT □ (0x00001000): Create in muted state - GST_PULSE_STREAM_START_MUTED □ (0x00002000): Try to adjust the latency of the sink/source based on the requested buffer metrics and adjust buffer metrics accordingly - GST_PULSE_STREAM_ADJUST_LATENCY □ (0x00004000): Enable compatibility mode for legacy clients that rely on a classic hardware device fragment-style playback model - GST_PULSE_STREAM_EARLY_REQUESTS □ (0x00008000): If set this stream won't be considered when it is checked whether the device this stream is connected to should auto-suspend - GST_PULSE_STREAM_DONT_INHIBIT_AUTO_SUSPEND □ (0x00010000): Create in unmuted state - GST_PULSE_STREAM_START_UNMUTED

Table 3-7 Element properties of pulsesrc (cont.)

Property	Description
	□ (0x00020000): If the sink/source this stream is connected to is suspended during the creation of this stream, cause it to fail - GST_PULSE_STREAM_FAIL_ON_SUSPEND □ (0x00040000): If a volume is passed when this stream is created, creation of this stream, cause it to fail - GST_PULSE_STREAM_RELATIVE_VOLUME □ ((0x00080000): Used to tag content that will be rendered by passthrough sinks - GST_PULSE_STREAM_PASSTHROUGH Write only

Usage

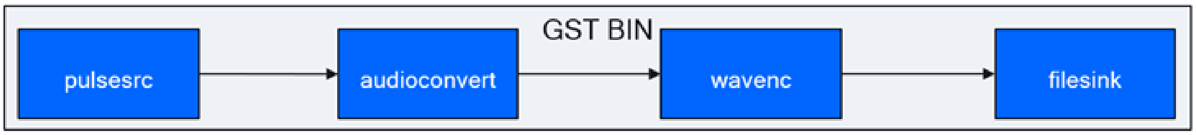


Figure 3-2 Audio capture with PCM encode

Command:

```
gst-launch-1.0 -v pulsesrc volume=10 ! audioconvert ! wavenc ! filesink
location=/data/track.wav
```

3.3 qtivtransform

The qtivtransform element leverages GPU hardware to upscale/downscale, flip, rotate, crop, and color covert incoming the YUV or RGB video frames determined by the element properties.

The exact X and Y axes placement with upscale/downscale for each input frame within the output can be set by the destination property with optional crop source set by the crop property. In addition to those main transformation the frame can be rotated (rotate property) at right angle increments and flipped (flip-horizontal and flip-vertical) horizontally and/or vertically.

If source pad GstCaps have not been specified and no transformation property set, the plug-in will try to negotiate the same capabilities as on the input. In this case the plug-in will operate in passthrough mode until some parameter changes.

Under the hood the plug-in uses QTI C2D library exposed by the Adreno for all transformation operations. This library is wrapped inside the custom GstC2dVideoConverter abstraction layer with APIs to create, configure and process the incoming and outgoing buffers. The output buffers are allocated by a custom buffer pool class called GstImageBufferPool which can allocate either GBM or ION buffers depending on the negotiated capabilities between this and downstream plug-ins. The GBM allocation is done via the QTI libgbm and ION allocation is done through IOCTL commands to the kernel.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstVideoTransform

qtivtransform pad configuration

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	video/x-raw	format:	{ (string)NV12, (string)NV21, (string)YUY2, (string)P010_10LE, (string)NV12_10LE32, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]
	video/x-raw(memory:GBM)	format:	{ (string)NV12, (string)NV21, (string)YUY2, (string)P010_10LE, (string)NV12_10LE32, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-raw	format:	{ (string)NV12, (string)NV21, (string)YUY2, (string)P010_10LE, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]
	video/x-raw(memory:GBM)	format:	{ (string)NV12, (string)NV21, (string)YUY2, (string)P010_10LE, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]

qtivtransform element configuration

Table 3-8 Element properties of qtivtransform

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "videotransform0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
flip-horizontal	■ Flip video image horizontally ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Boolean. Default: false
flip-vertical	■ Flip video image vertically ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Boolean. Default: false
rotate	■ Rotate video image ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Enum "GstVideoTransformRotate" Default: 0, "none" ■ (0): none - No rotation ■ (1): 90CW - Rotate 90 degrees clockwise ■ (2): 90CCW - Rotate 90 degrees counter-clockwise ■ (3): 180 - Rotate 180 degrees
crop	■ The crop rectangle inside the input ('<X, Y, WIDTH, HEIGHT >') ■ This property cannot be time synchronized. ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ GstValueArray of GValues of type "gint"
destination	■ Destination rectangle inside the output ('<X, Y, WIDTH, HEIGHT >') ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ GstValueArray of GValues of type "gint"
background	■ Background color ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4286611584
qos	■ Handle Quality-of-Service events ■ flags: readable, writable ■ Boolean. Default: false

Usage

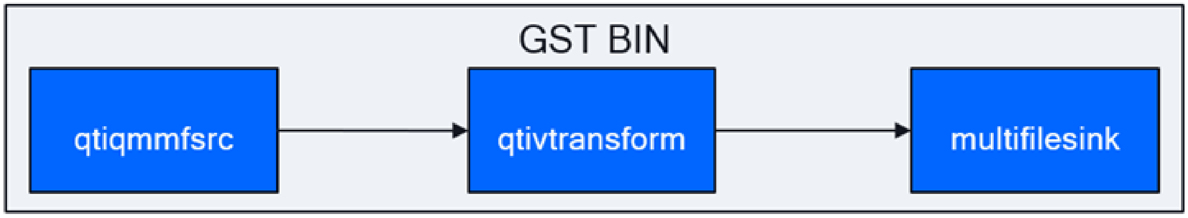


Figure 3-3 Convert and downscale YUV to RGB

Command:

```
gst-launch-1.0 -e qtiqmmfsrc ! video/x-raw\(memory:GBM
\),width=1920,height=1080,format=NV12,framerate=5/1 ! queue ! \
qtivtransform ! video/x-raw,width=1280,height=720,format=RGB ! multifilesink
location=/data/frame_%d.rgb
```

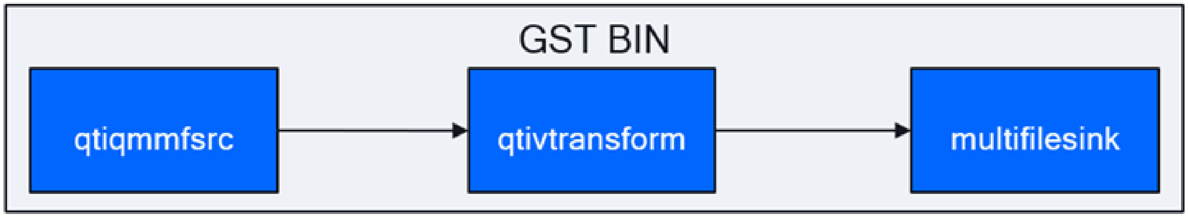


Figure 3-4 Flip and rotate YUV without color conversion and upscale/downscale

Command:

```
gst-launch-1.0 -e qtiqmmfsrc ! video/x-raw\(memory:GBM
\),width=1920,height=1080,format=NV12,framerate=5/1 ! queue ! \
qtivtransform rotate=90CW flip-horizontal=true ! multifilesink location=/data/
frame_%d.yuv
```

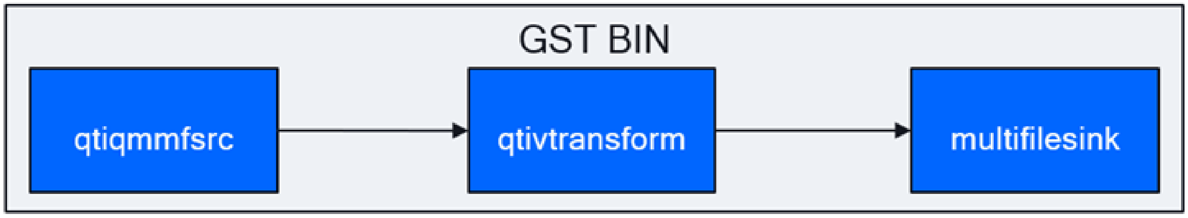


Figure 3-5 Center crop and upscale YUV to RGB

Command:

```
gst-launch-1.0 -e qtiqmmfsrc ! video/x-raw\(memory:GBM
\),width=1280,height=720,format=NV12,framerate=5/1 ! queue ! \
qtivtransform crop="<320,180,640,360>" ! video/x-
raw,width=1920,height=1080,format=RGB ! multifilesink location=/data/frames_
%d.rgb
```

3.4 qtivcomposer

The qtivcomposer element leverages the GPU hardware to merge/mix multiple input video streams into single output stream.

The precise composition of the input buffers (for each stream) within the output buffer is determined by the pad properties. The exact X and Y axis placement of each input frame within the output can be set by the position property with optional crop source set by the **crop** property while any upscale/ downscale is specified with the dimensions property.

In addition to those main transformation the frame can be rotated (rotate property) at right angle increments and flipped (flip-horizontal & flip-vertical) horizontally and/or vertically with additional alpha blending value (alpha property). By default, the frames are mixed in the order the sink pads were created, e.g., sink_0, sink_1, etc. The z order property can be used to change that order

Compositions include, but are not limited to: Picture-in-Picture, Side-by-Side, Alpha Blending, etc.

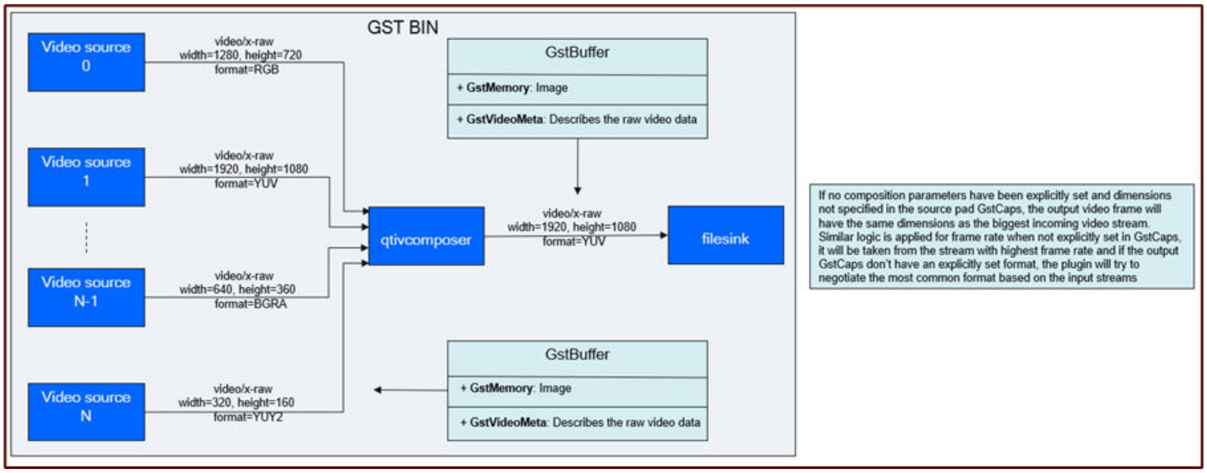
If no composition parameters have been explicitly set and dimensions not specified in source pad GstCaps, the output video frames will have the same dimensions as the biggest incoming video stream. Similar logic is applied for frame rate when not explicitly set in GstCaps, it will be taken from the stream with highest frame rate. And if output GstCaps don't have an explicitly set format the plug-in will try to negotiate the most common format based on the input streams.

Under the hood the plug-in uses QTI C2D library exposed by the Adreno for all composition operations. This library is wrapped inside the custom GstC2dVideoConverter abstraction layer with APIs to create, configure and process the incoming and outgoing buffers.

The output buffers are allocated by a custom buffer pool class called GstImageBufferPool which can allocate either GBM or ION buffers depending on the negotiated capabilities between this and downstream plug-ins. The GBM allocation is done through the QTI libgbm and ION allocation is done through IOCTL commands to the kernel.

Inheritance chain: GObject → GstObject → GstElement → GstAggregator → GstVideoComposer

Figure 3-6 gstpipeline with qtivtransform


GST BIN
Video source 0
video/x-raw width=1280, height=720 format=RGB
Video source 1
video/x-raw width=1920, height=1080 format=YUV
Video source N-1
video/x-raw width=640, height=360 format=BGRA
Video source N
video/x-raw width=320, height=160 format=YUY2
GstBuffer
+ GstMemory: Image
+ GstVideoMeta: Describes the raw video data
qtivcomposer
video/x-raw width=1920, height=1080 format=YUV
filesink
GstBuffer
+ GstMemory: Image
+ GstVideoMeta: Describes the raw video data

If no composition parameters have been explicitly set and dimensions not specified in the source pad GstCaps, the output video frame will have the same dimensions as the biggest incoming video stream. Similar logic is applied for frame rate when not explicitly set in GstCaps, it will be taken from the stream with highest frame rate and if the output GstCaps don't have an explicitly set format, the plugin will try to negotiate the most common format based on the input streams

qtivcomposer pad configuration

Table 3-9 Pad templates for qtivcomposer

Pad Name	Capabilities		
SINK template: 'sink_%u' *Availability:* On request *Direction:* sink	video/x-raw	format:	{ (string)NV12, (string)NV21, (string)UYVY, (string)YUY2, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]
	video/x-raw(memory:GBM)	format:	{ (string)NV12, (string)NV21, (string)UYVY, (string)YUY2, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-raw	format:	{ (string)NV12, (string)NV21, (string)UYVY, (string)YUY2, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]
	video/x-raw(memory:GBM)	format:	{ (string)NV12, (string)NV21, (string)UYVY, (string)YUY2, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]

Table 3-10 Pad properties for qtivcomposer

Pad	Property	Description
'sink_ %u'	zorder	Z axis order, default will be order of creation flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state, 0x40000000 Integer. Range: -1 - 2147483647 Default: -1
	crop	The crop rectangle ('<X, Y, WIDTH, HEIGHT >') flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state, 0x40000000 GstValueArray of GValues of type "gint"
	position	The X and Y coordinates of the destination rectangle top left corner ('<X, Y>') flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state, 0x40000000 GstValueArray of GValues of type "gint"
	dimensions	The destination rectangle width and height, if left as '0' they will be the same as input dimensions ('<WIDTH, HEIGHT>') flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state, 0x40000000 GstValueArray of GValues of type "gint"
	alpha	Alpha channel value flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state, 0x40000000 Double. Range: 0.0 - 1.0 Default: 1.0
	flip-horizontal	Flip video image horizontally flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state, 0x40000000 Boolean. Default: false
	flip-vertical	Flip video image vertically flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state, 0x40000000 Boolean. Default: false
	rotate	Rotate video image flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state, 0x40000000 Enum "GstVideoComposerRotate" Default: 0, "none" ■ (0): none - No rotation ■ (1): 90CW - Rotate 90 degrees clockwise ■ (2): 90CCW - Rotate 90 degrees counter-clockwise ■ (3): 180 - Rotate 180 degrees
	emit-signals	Send signals to signal data consumption flags: readable, writable Boolean. Default: false

qtivcomposer element configuration

Table 3-11 Element properties for qtivcomposer

Property	Description
name	The name of the object flags: readable, writable String. Default: "videocomposer0"
parent	The parent of the object flags: readable, writable Object of type "GstObject"
latency	Additional latency in live mode to allow upstream to take longer to produce buffers for the current position (in nanoseconds) flags: readable, writable Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
start-time-selection	Decides which start time is output flags: readable, writable Enum "GstAggregatorStartTimeSelection" Default: 0, "zero" ■ (0): zero - Start at 0 running time (default) ■ (1): first - Start at first observed input running time ■ (2): set - Set start time with start-time property
start-time	Start time to use if start-time-selection=set flags: readable, writable Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 18446744073709551615
background	Background color flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state Unsigned Integer. Range: 0 - 4294967295 Default: 4286611584
min-upstream-latency	When sources with a higher latency are expected to be plugged in dynamically after the aggregator has started playing, this allows overriding the minimum latency reported by the initial source(s). This is only considered when larger than the actually reported minimum latency. (nanoseconds) flags: readable, writable Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0

Usage

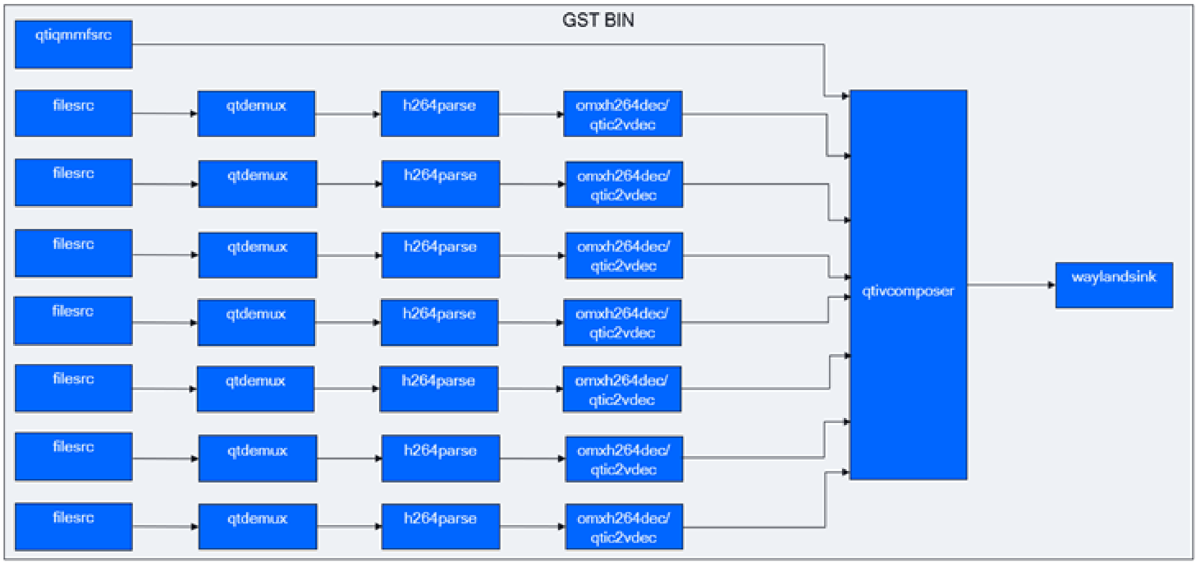


Figure 3-7 Eight video stream composition

Run the prerequisites to start the Weston server:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer \
sink_0::position="<0, 0>" sink_0::dimensions="<640, 360>" \
sink_1::position="<640, 0>" sink_1::dimensions="<640, 360>" \
sink_2::position="<1280, 0>" sink_2::dimensions="<640, 360>" \
sink_3::position="<0, 360>" sink_3::dimensions="<640, 360>" \
sink_4::position="<1280, 360>" sink_4::dimensions="<640, 360>" \
sink_5::position="<0, 720>" sink_5::dimensions="<640, 360>" \
sink_6::position="<640, 720>" sink_6::dimensions="<640, 360>" \
sink_7::position="<1280, 720>" sink_7::dimensions="<640, 360>" \
mixer. ! video/x-raw\(memory:GBM\),format=NV12,compression=ubwc ! queue !
waylandsink sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM\),width=1280,height=720 !
queue ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! h264parse !
qtic2vdec ! queue ! mixer. \
filesrc location=/data/Driving_720p_180s_30FPS.MOV ! qtdemux ! h264parse !
qtic2vdec ! queue ! mixer. \
filesrc location=/data/Street_Side_720p_180s_30FPS.MOV ! qtdemux !
h264parse !  qtic2vdec ! queue ! mixer. \
filesrc location=/data/Street_Bridge_720p_180s_30FPS.MOV ! qtdemux !
h264parse !  qtic2vdec ! queue ! mixer. \
filesrc location=/data/Carview_720p_180s_30FPS.MOV ! qtdemux ! h264parse !
qtic2vdec ! queue ! mixer. \
filesrc location=/data/Animals_000_720p_180s_30FPS.mp4 ! qtdemux !
h264parse !  qtic2vdec ! queue ! mixer. \
filesrc location=/data/Animals_002_720p_180s_30FPS.mp4 ! qtdemux !
h264parse !  qtic2vdec ! queue ! mixer.
```

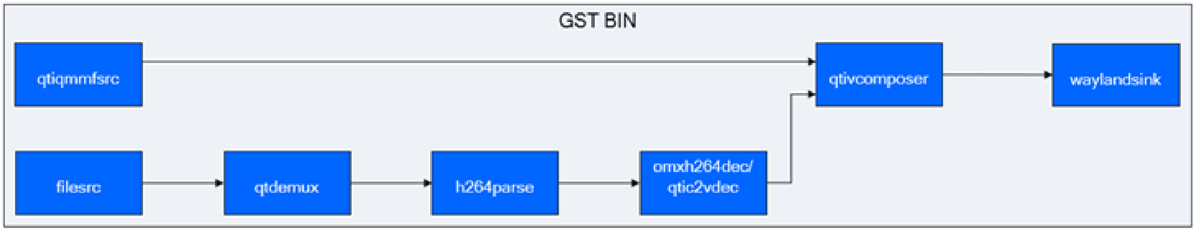


Figure 3-8 Two video stream composition

Run the prerequisites to start the Weston server:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_0::position="<0, 0>" sink_0::dimensions="<1920, 1080>" \
sink_1::position="<1280, 720>" sink_1::dimensions="<640, 360>" \ ! queue !
waylandsink sync=false fullscreen=true (No space here)
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM\),width=1920,height=1080 !
queue ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! h264parse !
qtic2vdec ! queue ! mixer
```

3.5 qtivsplit

The qtivsplit element leverages GPU hardware to "split" a single input video stream into multiple output streams whose number is a user defined variable and is equal to the number of source pads.

The precise method for "splitting" that incoming stream is determined by the mode property. In 'normal' operational mode, incoming video frames on the sink pad are duplicated on each of the source pads with additional color conversion and upscale/downscale applied based on the negotiated GstCaps on that pad.

While when operating in 'roi' mode, input video buffers are checked for GstVideoRegionOfInterestMeta. Each such meta entry in that GstBuffer is sent to its corresponding source pad based on the 'id' field and additionally crop, upscale/downscale and color conversion is performed in order to match the negotiated GstCaps.

Source pads with no corresponding ROI meta will produce GAP buffers.

Care must be taken in 'roi' mode when source pads are created so that their number is equal to the maximum expected count of GstVideoRegionOfInterestMeta inside a single GstBuffer. If their source pads are fewer, the ROI metas that do not have a corresponding pad will be ignored.

Under the hood the plug-in uses either QTI Adreno C2D or QTI IB2C library for all transformation operations. This library is wrapped inside the custom GstC2dVideoConverter or GstGlesVideoConverter abstraction layer respectively with APIs to create, configure and process the incoming and outgoing buffers.

he output buffers are allocated by a custom buffer pool class called GstImageBufferPool, which can allocate either GBM or ION buffers depending on the negotiated capabilities between this and downstream plug-ins. The GBM allocation is done via the QTI libgbm and ION allocation is done through IOCTL commands to the kernel.

Inheritance chain: GObject → GstObject → GstElement → GstVideoSplit

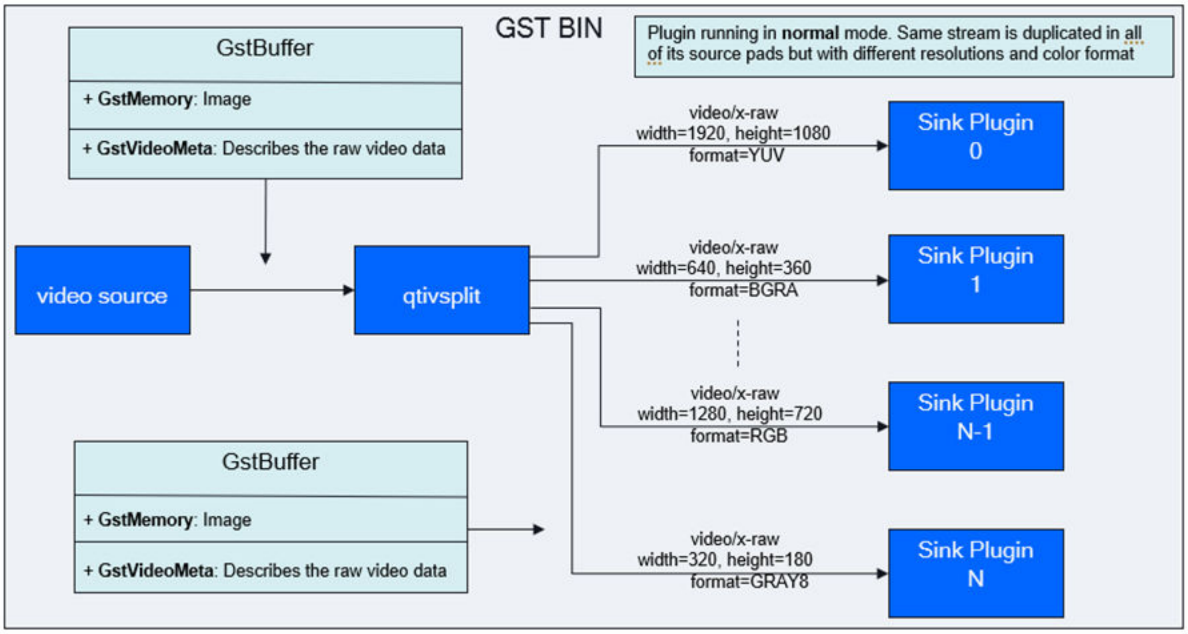


Figure 3-9 Gstreamer pipeline with qtivsplit

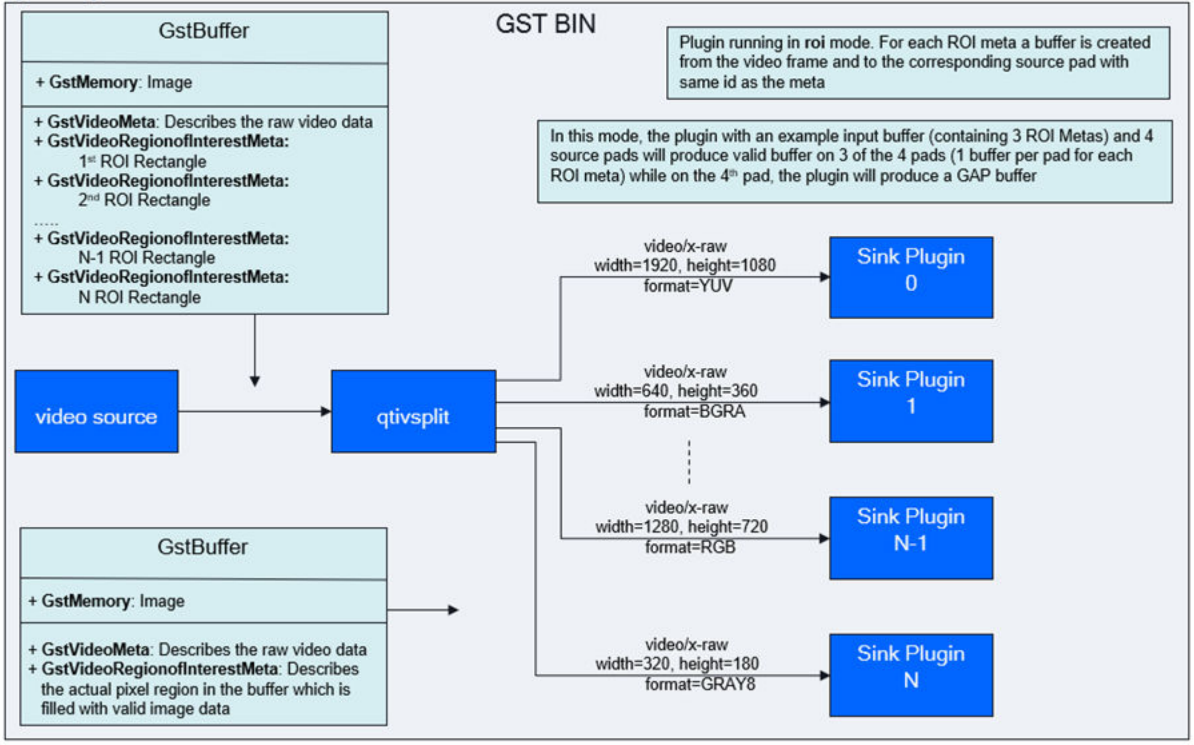

GST BIN
Plugin running in roi mode. For each ROI meta a buffer is created from the video frame and to the corresponding source pad with same id as the meta
In this mode, the plugin with an example input buffer (containing 3 ROI Metas) and 4 source pads will produce valid buffer on 3 of the 4 pads (1 buffer per pad for each ROI meta) while on the 4th pad, the plugin will produce a GAP buffer
GstBuffer
+ GstMemory: Image
+ GstVideoMeta: Describes the raw video data
+ GstVideoRegionofInterestMeta: 1st ROI Rectangle
+ GstVideoRegionofInterestMeta: 2nd ROI Rectangle
.....
+ GstVideoRegionofInterestMeta: N-1 ROI Rectangle
+ GstVideoRegionofInterestMeta: N ROI Rectangle
video source
qtivsplit
video/x-raw
width=1920, height=1080
format=YUV
Sink Plugin 0
video/x-raw
width=640, height=360
format=BGRA
Sink Plugin 1
video/x-raw
width=1280, height=720
format=RGB
Sink Plugin N-1
video/x-raw
width=320, height=180
format=GRAY8
Sink Plugin N
GstBuffer
+ GstMemory: Image
+ GstVideoMeta: Describes the raw video data
+ GstVideoRegionofInterestMeta: Describes the actual pixel region in the buffer which is filled with valid image data

Figure 3-10 qtivsplit - buffer management

qtivsplit pad configuration

Table 3-12 Pad templates for qtivsplit

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	video/x-raw	format:	{ (string)NV12, (string)NV21, (string)UYVY, (string)YUY2, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]
	video/x-raw(memory:GBM)	format:	{ (string)NV12, (string)NV21, (string)UYVY, (string)YUY2, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]
SRC template: 'src_%u' *Availability:* On request *Direction:* source	video/x-raw	format:	{ (string)NV12, (string)NV21, (string)UYVY, (string)YUY2, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]
	video/x-raw(memory:GBM)	format:	{ (string)NV12, (string)NV21, (string)UYVY, (string)YUY2, (string)RGBA, (string)BGRA, (string)ARGB, (string)ABGR, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)RGB, (string)BGR, (string)GRAY8 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 255/1]

qtivsplit element configuration

Table 3-13 Element properties of qtivsplit

Property	Description
name	▪ The name of the object ▪ flags: readable, writable ▪ String. Default: "videosplit0"
parent	▪ The parent of the object ▪ flags: readable, writable ▪ Object of type "GstObject"
mode	▪ Operational mode ▪ flags: readable, writable, changeable only in NULL or READY state Enum "GstVideoSplitMode" Default: 0, "normal" ▪ (0): normal - Normal mode. Incoming buffer is rescaled, and color converted for each of the source pads in order to match the negotiated caps. ▪ (1): roi - ROI mode. Incoming buffer is checked for ROI meta. For each meta entry a crop, rescale and color conversion are performed, and then sent to the corresponding source pad. Pads with no corresponding ROI meta will produce GAP buffers.

Usage

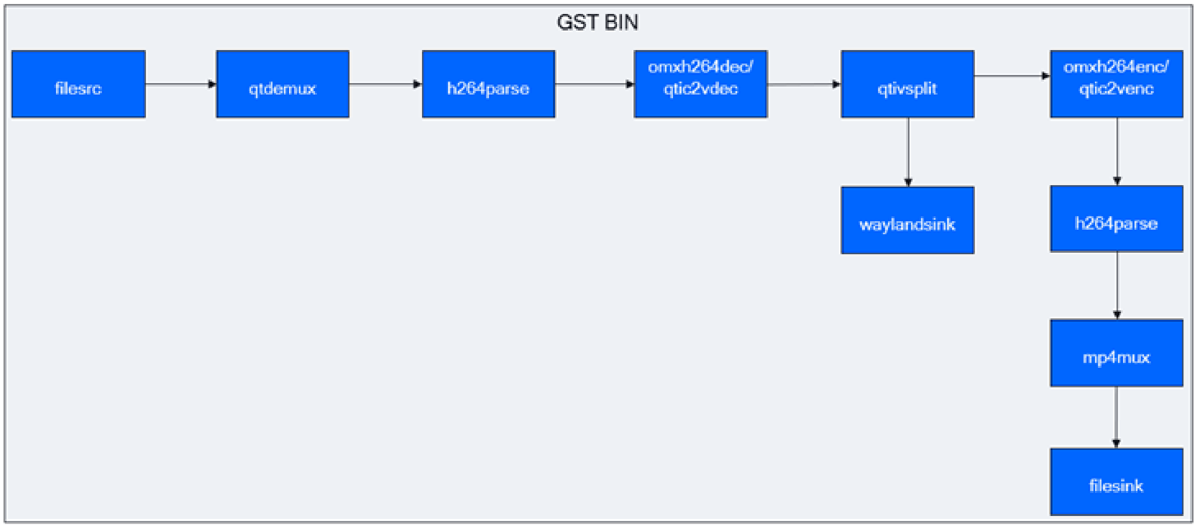


Figure 3-11 Video encode and display from a video stream

Run prerequisites to start the Weston server:

```
gst-launch-1.0 -e --gst-debug=2 \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! queue ! qtivsplit name=vsplit \
vsplit. ! video/x-raw\(memory:GBM\),width=640,height=360,format=NV12 !
queue ! qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/
data/video.mp4" \
vsplit. ! video/x-raw\(memory:GBM
\),width=1280,height=720,format=NV12,compression=ubwc ! queue ! waylandsink
fullscreen=true sync=true
```

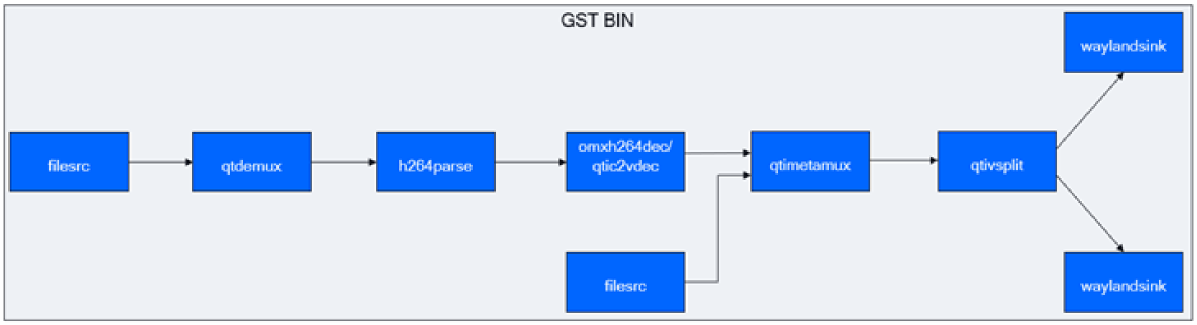


Figure 3-12 Take ROIs from file and split them into multiple streams

Run prerequisites to start the Weston server:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue ! qtivsplit
name=vsplit src_0::mode=single-roi-meta src_1::mode=single-roi-meta \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! queue ! metamux. \
filesrc location=/data/roi.bin ! text/x-raw ! queue ! metamux. \
vsplit. ! video/x-raw\(memory:GBM
\),width=640,height=360,format=NV12,compression=ubwc ! queue ! waylandsink
x=0 y=0 width=640 height=360 sync=true \
vsplit. ! video/x-raw\(memory:GBM
\),width=640,height=360,format=NV12,compression=ubwc ! queue ! waylandsink
x=1280 y=0 width=640 height=360 sync=true
```

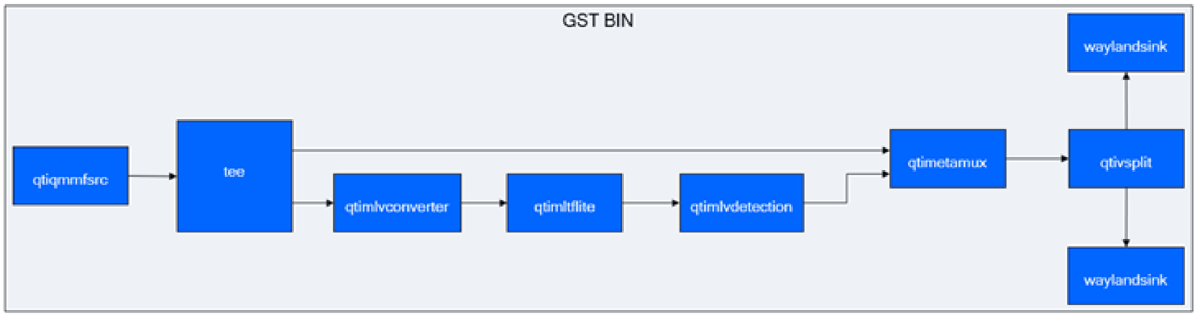


Figure 3-13 Take ROIs from ML inference and split them into multiple streams

Run prerequisites to start the Weston server:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue ! qtivsplit
name=vsplit src_0::mode=single-roi-meta src_1::mode=single-roi-meta \
qtiqmmfsrc video/x-raw\(memory:GBM\),width=1920,height=1080,format=NV12 !
queue ! tee name=t_split ! queue ! metamux. \
t_split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection
threshold=75.0 results=2 module=yolov5m labels=/data/yolov5m.labels ! text/x-
raw ! queue ! metamux. \
vsplit. ! video/x-raw\(memory:GBM
\),width=640,height=360,format=NV12,compression=ubwc ! queue ! waylandsink
x=0 y=0 width=640 height=360 sync=true \
vsplit. ! video/x-raw\(memory:GBM
\),width=640,height=360,format=NV12,compression=ubwc ! queue ! waylandsink
x=1280 y=0 width=640 height=360 sync=true
```

3.6 qtioverlay

The qtioverlay element is a hardware accelerated in-place image draw and blit plug-in for drawing overlays on top of YUV images.

NOTE This plug-in will be replaced during a future release.

The overlays can be configured through properties and also through buffer metadata attached to the input buffers. It supports blitting of static images, bounding boxes, custom user text, data/time overlays, privacy mask through properties. And supports blitting of detection (bounding box), segmentation (semantic mask/mask image), classification (label/user text), and pose graph overlay through metadata attached to the input buffer.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstVideoFliter → GstVideoOverlay

qtioverlay pad configuration

Table 3-14 Pad templates for qtioverlay

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	video/x-raw	format: width: height: framerate:	{ (string)NV12, (string)NV21 } [1, 32767] [1, 32767] [0/1, 2147483647/1]
	video/x-raw(ANY)	format: width: height: framerate:	{ (string)NV12, (string)NV21 } [1, 32767] [1, 32767] [0/1, 2147483647/1]
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-raw	format: width: height: framerate:	{ (string)NV12, (string)NV21 } [1, 32767] [1, 32767] [0/1, 2147483647/1]
	video/x-raw(ANY)	format: width: height: framerate:	{ (string)NV12, (string)NV21 } [1, 32767] [1, 32767] [0/1, 2147483647/1]

qtioverlay element configuration

Table 3-15 Element properties for qtioverlay

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "overlay0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
qos	■ Handle Quality-of-Service events ■ flags: readable, writable ■ Boolean. Default: true
overlay-text	■ Renders text on top of video stream ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: ""
overlay-date	■ Renders date and time on top of video stream ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: ""
overlay-simg	■ Renders static image on top of video stream ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: ""
overlay-bbox	■ Renders bounding box and label on top of video stream ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: ""
overlay-mask	■ Renders privacy mask on top of video stream ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ String. Default: ""
bbox-color	■ Bounding box overlay color ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 52479
date-color	■ Date overlay color ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4278190335
text-color	■ Text overlay color ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4294902015
pose-color	■ Pose overlay color ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 869007615

Table 3-15 Element properties for qtioverlay (cont.)

Property	Description
arrows-color	■ Arrows overlay color ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4278190335
bbox-font-size	■ Bounding box overlay font size ■ flags: readable, writable ■ Unsigned Integer. Range: 1 - 4294967295 Default: 25
date-font-size	■ Date overlay font size ■ flags: readable, writable ■ Unsigned Integer. Range: 1 - 4294967295 Default: 20
text-font-size	■ Text overlay font size ■ flags: readable, writable ■ Unsigned Integer. Range: 1 - 4294967295 Default: 40
dest-rect-ml-text	■ Destination rectangle params for ML Detection overlay. The Start-X, Start-Y , Width, Height of the destination rectangle format is <X, Y, WIDTH, HEIGHT> ■ flags: readable, writable ■ GstValueArray of GValues of type "gint"
arrows-ft-mv	■ Arrows mv filter ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
arrows-ft-sad	■ Arrows sad filter ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
arrows-ft-var	■ Arrows var filter ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0

Usage

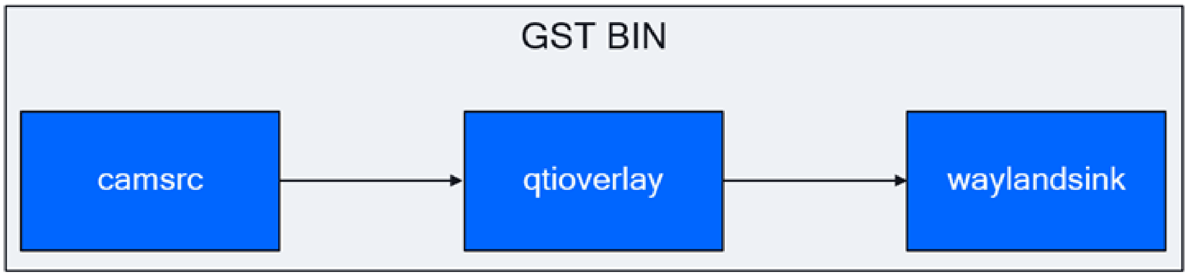


Figure 3-14 User text overlay

Command:

```
setprop persist.overlay.use_c2d_blit 2 && export XDG_RUNTIME_DIR=/run/user/
root && gst-launch-1.0 qtiqmmfsrc ! video/x-raw\(memory:GBM\), format=NV12,
width=1920, height=1080, framerate=30/1 ! qtioverlay overlay-text="text0,
text=\"Qualcomm\ Intelligence\", color=(uint)0xFFFF00FF, dest-rect=<160, 624,
944, 50>;" ! waylandsink fullscreen=true
```

3.7 qtimetamux

The qtimetamux element uses frame matching technics to associate/attach ML string based postprocessing results (output from postprocessing plug-in) or CV information to original frame as GstMeta.

This output can be fed to plug-ins such as qtioverlay for rendering or can be passed down the ML inference pipeline for next level inferencing. It is also possible to attach an appsink plug-in and received the GstBuffer along with the GstMeta in user defined application.

Inheritance chain: GObject → GstObject → GstElement → GstMetaMux

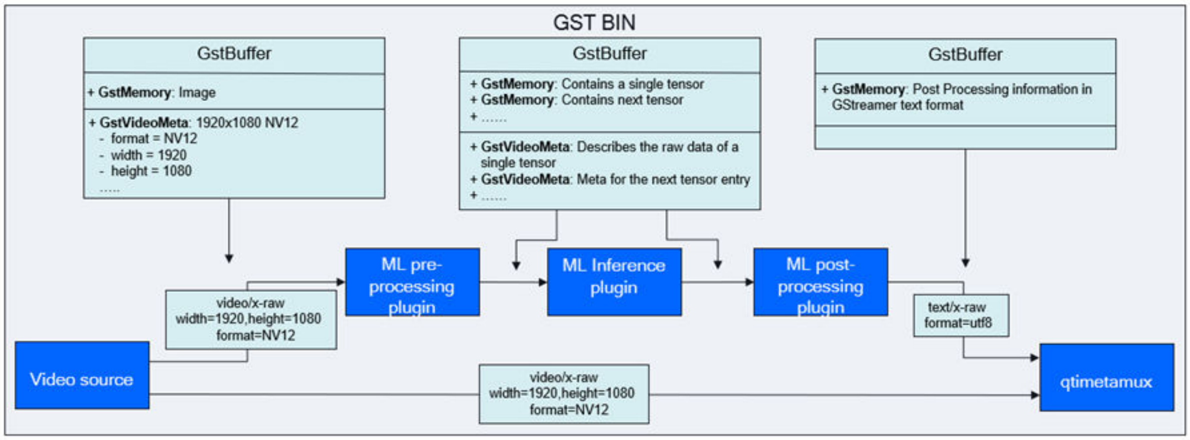

GST BIN
GstBuffer
+ GstMemory: Image
+ GstVideoMeta: 1920x1080 NV12
- format = NV12
- width = 1920
- height = 1080
.....
GstBuffer
+ GstMemory: Contains a single tensor
+ GstMemory: Contains next tensor
+ ......
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ......
GstBuffer
+ GstMemory: Post Processing information in GStreamer text format
ML pre-processing plugin
ML Inference plugin
ML post-processing plugin
video/x-raw
width=1920,height=1080
format=NV12
text/x-raw
format=utf8
Video source
video/x-raw
width=1920,height=1080
format=NV12
qtimetamux

Figure 3-15 Metamux ML ROI with video

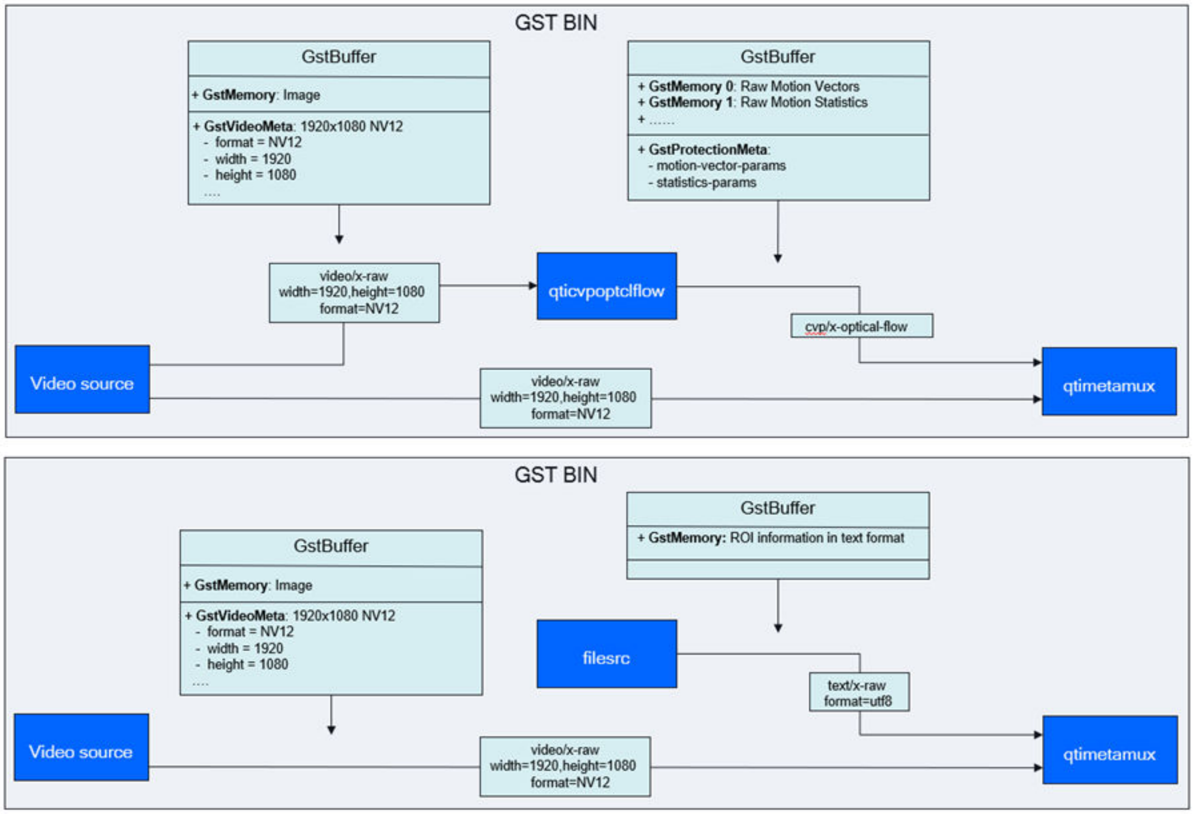


Figure 3-16 Metamux filesrc ROI with video

qtimetamux pad configuration

Table 3-16 Pad templates for qtimetamux

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	video/x-raw(ANY)	–	–
	audio/x-raw(ANY)	–	–
SINK template: 'data_%u' *Availability:* On request *Direction:* sink	text/x-raw	format:	utf8
	cv/x-optical-flow	–	–
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-raw(ANY)	–	–
	audio/x-raw(ANY)	–	–

qtimetamux element configuration

Table 3-17 Element properties of qtimetamux

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "metamux0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
latency	■ Additional latency to allow more time for upstream to produce metadata entries for the current position (in nanoseconds). ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
mode	■ Operational mode ■ flags: readable, writable, changeable only in NULL or READY state ■ Enum "GstMetaMuxMode" Default: 0, "async" ■ (0): async - No timestamp synchronization is done between the media buffers and the incoming metadata entries. When a media buffer arrives, it will wait until there are metadata entries on all data pads. ■ (1): sync - Timestamp matching between media buffers & metadata entries is enabled. When a media buffer arrives it will wait a maximum of '1 / framerate' (for video caps) or '1 / rate' (for audio caps) time to receive meta entries on all pads with timestamps matching that of the buffer.

Usage

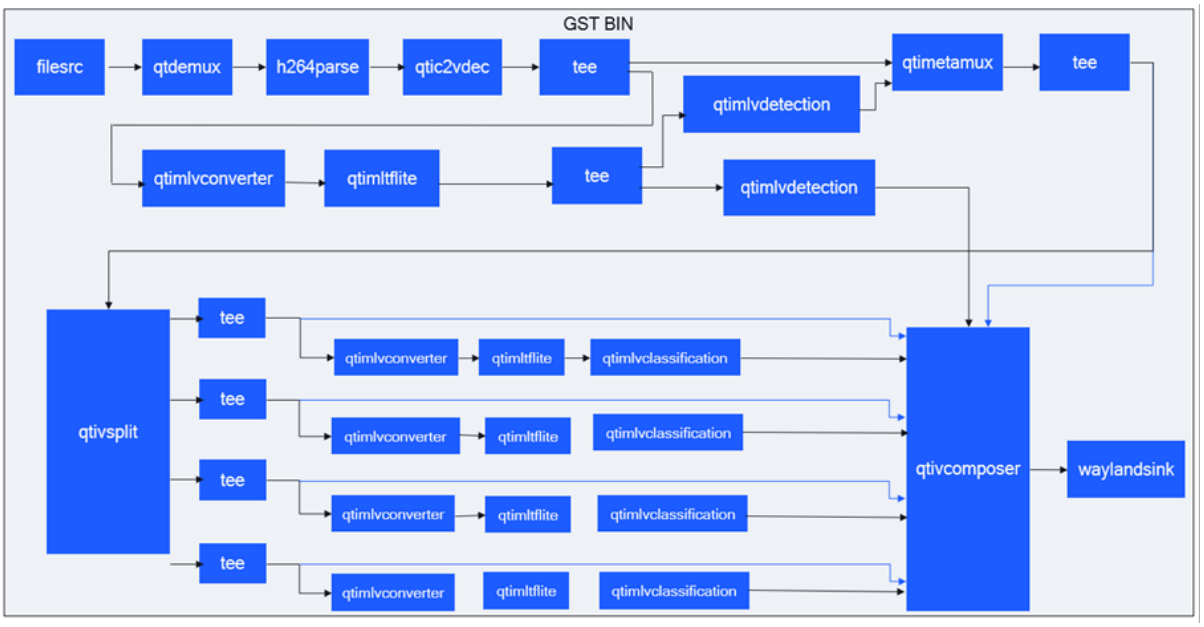


Figure 3-17 Daisy chaining ML object detection with image classification

Command:

```
ulimit -n 4096 && gst-launch-1.0 -e --gst-debug=2,fpsdisplaysink:6 \
qtimltflite name=TFLite_yolov5 delegate=hexagon model=/data/yolov5m-320x320-
int8.tflite \
qtimltflite name=TFLite_MobileNet_1 delegate=hexagon model=/data/
mobilenet_v2_1.0_224_quant.tflite \
qtimltflite name=TFLite_MobileNet_2 delegate=hexagon model=/data/
mobilenet_v2_1.0_224_quant.tflite \
qtimltflite name=TFLite_MobileNet_3 delegate=hexagon model=/data/
mobilenet_v2_1.0_224_quant.tflite \
qtimltflite name=TFLite_MobileNet_4 delegate=hexagon model=/data/
mobilenet_v2_1.0_224_quant.tflite \
qtimlvconverter name=ml_convert_0 ! queue max-size-bytes=0 max-size-time=0 !
TFLite_yolov5. TFLite_yolov5. ! queue max-size-bytes=0 max-size-time=0 ! tee
name=t_split_1 \
qtivcomposer name=mixer \
sink_0::position="<0, 0>" sink_0::dimensions="<1280, 720>" \
sink_1::position="<0, 0>" sink_1::dimensions="<1280, 720>" \
sink_2::position="<0, 0>" sink_2::dimensions="<384, 216>" \
sink_3::position="<896, 0>" sink_3::dimensions="<384, 216>" \
sink_4::position="<0, 504>" sink_4::dimensions="<384, 216>" \
sink_5::position="<896, 504>" sink_5::dimensions="<384, 216>" \
sink_6::position="<0, 0>" sink_6::dimensions="<384, 40>" \
sink_7::position="<896, 0>" sink_7::dimensions="<384, 40>" \
sink_8::position="<0, 504>" sink_8::dimensions="<384, 40>" \
sink_9::position="<896, 504>" sink_9::dimensions="<384, 40>" \
mixer. ! video/x-raw\(memory:GBM\),format=NV12,compression=ubwc ! queue max-
size-bytes=0 max-size-time=0 ! fpsdisplaysink sync=false signal-fps-
measurements=true text-overlay=false video-sink="waylandsink sync=false
fullscreen=true" \
filesrc location=/data/Street_Side_720p_180s_30FPS.MOV ! qtdemux !
h264parse ! qtivdec ! video/x-raw\(memory:GBM
\),format=NV12,compression=ubwc ! queue max-size-bytes=0 max-size-time=0 !
tee name=v_split_1 ! queue max-size-bytes=0 max-size-time=0 ! metamux1.
v_split_1. ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_0. \
t_split_1. ! queue max-size-bytes=0 max-size-time=0 ! qtimlvdetection
threshold=75.0 results=4 module=yolov5m labels=/data/yolov5m.labels ! text/x-
raw ! queue max-size-bytes=0 max-size-time=0 ! qtimetamux name=metamux1 !
queue max-size-bytes=0 max-size-time=0 ! tee name=t_split_2 ! queue max-size-
bytes=0 max-size-time=0 ! mixer. \
t_split_1. ! queue max-size-bytes=0 max-size-time=0 ! qtimlvdetection
threshold=75.0 results=5 module=yolov5m labels=/data/yolov5m.labels ! video/x-
raw,width=512,height=288 ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
t_split_2. ! queue max-size-bytes=0 max-size-time=0 ! qtivsplit name=vsplit1
mode=roi \
vsplit1. ! video/x-raw,format=BGRA ! queue max-size-bytes=0 max-size-time=0 !
tee name=split_1 ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_1.
split_1. ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
vsplit1. ! video/x-raw,format=BGRA ! queue max-size-bytes=0 max-size-time=0 !
tee name=split_2 ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_2.
split_2. ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
vsplit1. ! video/x-raw,format=BGRA ! queue max-size-bytes=0 max-size-time=0 !
tee name=split_3 ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_3.
split_3. ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
vsplit1. ! video/x-raw,format=BGRA ! queue max-size-bytes=0 max-size-time=0 !
tee name=split_4 ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_4.
split_4. ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
qtimlvconverter name=ml_convert_1 ! queue max-size-bytes=0 max-size-time=0 !
TFLite_MobileNet_1. TFLite_MobileNet_1. ! queue max-size-bytes=0 max-size-
time=0 ! mlclass_1. \
qtimlvconverter name=ml_convert_2 ! queue max-size-bytes=0 max-size-time=0 !
TFLite_MobileNet_2. TFLite_MobileNet_2. ! queue max-size-bytes=0 max-size-
time=0 ! mlclass_2. \
qtimlvconverter name=ml_convert_3 ! queue max-size-bytes=0 max-size-time=0 !
TFLite_MobileNet_3. TFLite_MobileNet_3. ! queue max-size-bytes=0 max-size-
time=0 ! mlclass_3. \
qtimlvconverter name=ml_convert_4 ! queue max-size-bytes=0 max-size-time=0 !
TFLite_MobileNet_4. TFLite_MobileNet_4. ! queue max-size-bytes=0 max-size-
time=0 ! mlclass_4. \
qtimlvclassification name=mlclass_1 threshold=51.0 results=2 module=mobilenet
labels=/data/mobilenet.labels ! video/x-raw,width=384,height=40 ! queue max-
size-bytes=0 max-size-time=0 ! mixer. \
qtimlvclassification name=mlclass_2 threshold=51.0 results=2 module=mobilenet
labels=/data/mobilenet.labels ! video/x-raw,width=384,height=40 ! queue max-
size-bytes=0 max-size-time=0 ! mixer. \
qtimlvclassification name=mlclass_3 threshold=51.0 results=2 module=mobilenet
labels=/data/mobilenet.labels ! video/x-raw,width=384,height=40 ! queue max-
size-bytes=0 max-size-time=0 ! mixer. \
qtimlvclassification name=mlclass_4 threshold=51.0 results=2 module=mobilenet
labels=/data/mobilenet.labels ! video/x-raw,width=384,height=40 ! queue max-
size-bytes=0 max-size-time=0 ! mixer
```

3.8 qtisocketsrc

The qtisocketsrc element utilizes UNIX sockets to receive file descriptor backed GstBuffers from other processes that have qtisocketsink as an exit point.

The qtisocketsrc plug-in requires a socket file (should have the extension of .sock) to be passed as the socket property so it can receive the file descriptor.

After connecting to a socket, the plug-in polls/waits to receive a GstBuffer that it will use to create a block and fill it with the frame data received. Customizations on buffer such as size in bytes to be read per buffer is specified with blocksize, and number of buffers to output before sending EOF signal is controlled with num-buffers.

Inheritance chain: GObject → GstObject → GstElement → GstBaseSrc → GstPushSrc → GstFdSocketSrc

qtisocketsrc pad configuration

Table 3-18 Pad templates for qtisocketsrc

Pad Name	Capabilities	
SRC template: 'src' *Availability:* Always *Direction:* source	ANY	–

qtisocketsrc element configuration

Table 3-19 Element properties of qtisocketsrc

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "fdsocketsrc0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
blocksize	■ Size in bytes to read per buffer (-1 = default) ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4096
num-buffers	■ Number of buffers to output before sending EOS (-1 = unlimited) ■ flags: readable, writable ■ Integer. Range: -1 - 2147483647 Default: -1
typefind	■ Run typefind before negotiating (deprecated, non-functional) ■ flags: readable, writable, deprecated ■ Boolean. Default: false
do-timestamp	■ Apply current stream time to buffers ■ flags: readable, writable ■ Boolean. Default: false
socket	■ Location of the Unix Domain Socket ■ flags: readable, writable, changeable only in NULL or READY state ■ String. Default: null
timeout	■ Socket post timeout ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0

Usage

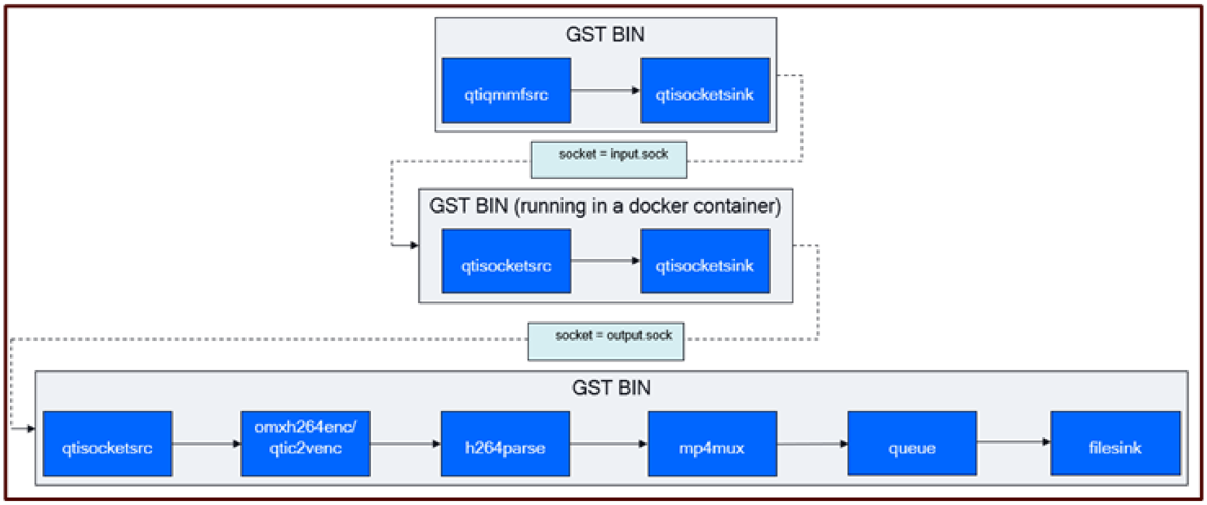


Figure 3-18 Transmitting single H264 encoded camera stream

Command:

```
# Start 3 separate adb shell consoles

# In 1st console:
gst-launch-1.0 -e --gst-debug=2,qtisocketsrc:6 qtisocketsrc socket=/tmp/
docker/output.sock ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080 ! omxh264enc target-bitrate=6000000
periodicity-idr=1 interval-intraframes=29 control-rate=max-bitrate !
h264parse ! mp4mux ! queue ! filesink location=/data/video.mp4

# In 2nd console:
docker run -v /tmp/docker:/tmp/docker -v /usr/lib/gstreamer-1.0/
libqtisocketsrc.so:/usr/lib/aarch64-linux-gnu/gstreamer-1.0/
libqtisocketsrc.so -v /usr/lib/gstreamer-1.0/libqtisocketsink.so:/usr/lib/
aarch64-linux-gnu/gstreamer-1.0/libqtisocketsink.so -ti gst-image /bin/bash -
c "gst-launch-1.0 -e --gst-debug=4 --gst-
debug=2,qtisocketsink:6,qtisocketsrc:6 qtisocketsrc socket=/tmp/docker/
input.sock ! qtisocketsink socket=/tmp/docker/output.sock"

# In 3rd console:
gst-launch-1.0 -e --gst-debug=2,qtisocketsink:6 qtiqmmfsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1,camera=0 !
qtisocketsink socket=/tmp/docker/input.sock
```

3.9 qtisocketsink

The qtisocketsink element utilizes UNIX sockets to transfer file descriptor backed GstBuffers to other processes with qtisocketsrc as entry point.

The qtisocketsink plug-in requires a UNIX domain socket file (should have the extension of .sock) to be passed as the *socket* property to transfer the file descriptor.

The buffers passed from the other process in to this plug-in are tracked with a reference counter, which is incremented or decremented accordingly when the buffers are sent or returned through the socket.

The *async* property can set to false for the sockets to transition asynchronously to PAUSED state. For more advance configuration, the maximum number of nanoseconds that a buffer can be late before it is dropped can be adjusted via the max-lateness property and the maximum bits per second to render can be set with the max-bitrate property.

Inheritance chain: GObject → GstObject → GstElement → GstBaseSink → GstFdSocketSink

qtisocketsink pad configuration

Table 3-20 Pad templates for qtisocketsink

Pad Name	Capabilities	
SINK template: 'sink' *Availability:* Always *Direction:* sink	ANY	

qtisocketsink element configuration

Table 3-21 Element properties for qtisocketsink

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "fdsocketsink0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
sync	■ Sync on the clock ■ flags: readable, writable ■ Boolean. Default: true
max-lateness	■ Maximum number of nanoseconds that a buffer can be late before it is dropped (-1 unlimited) ■ flags: readable, writable ■ Integer64. Range: -1 - 9223372036854775807 Default: -1
qos	■ Generate Quality-of-Service events upstream ■ flags: readable, writable ■ Boolean. Default: false
async	■ Go asynchronously to PAUSED ■ flags: readable, writable ■ Boolean. Default: true
ts-offset	Timestamp offset in nanoseconds ■ flags: readable, writable ■ Integer64. Range: -9223372036854775808 - 9223372036854775807 Default: 0

Table 3-21 Element properties for qtisocketsink (cont.)

Property	Description
enable-last-sample	▪ Enable the last-sample property ▪ flags: readable, writable ▪ Boolean. Default: true
last-sample	▪ The last sample received in the sink ▪ flags: readable ▪ Boxed pointer of type "GstSample"
blocksize	▪ Size in bytes to pull per buffer (0 = default) ▪ flags: readable, writable ▪ Unsigned Integer. Range: 0 - 4294967295 Default: 4096
render-delay	▪ Additional render delay of the sink in nanoseconds ▪ flags: readable, writable ▪ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
throttle-time	▪ The time to keep between rendered buffers (0 = disabled) ▪ flags: readable, writable ▪ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
max-bitrate	▪ The maximum bits per second to render (0 = disabled) ▪ flags: readable, writable ▪ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
socket	▪ Location of the Unix Domain Socket ▪ flags: readable, writable, changeable only in NULL or READY state ▪ String. Default: null
processing-deadline	▪ Maximum processing deadline in nanoseconds ▪ flags: readable, writable ▪ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 20000000

Usage

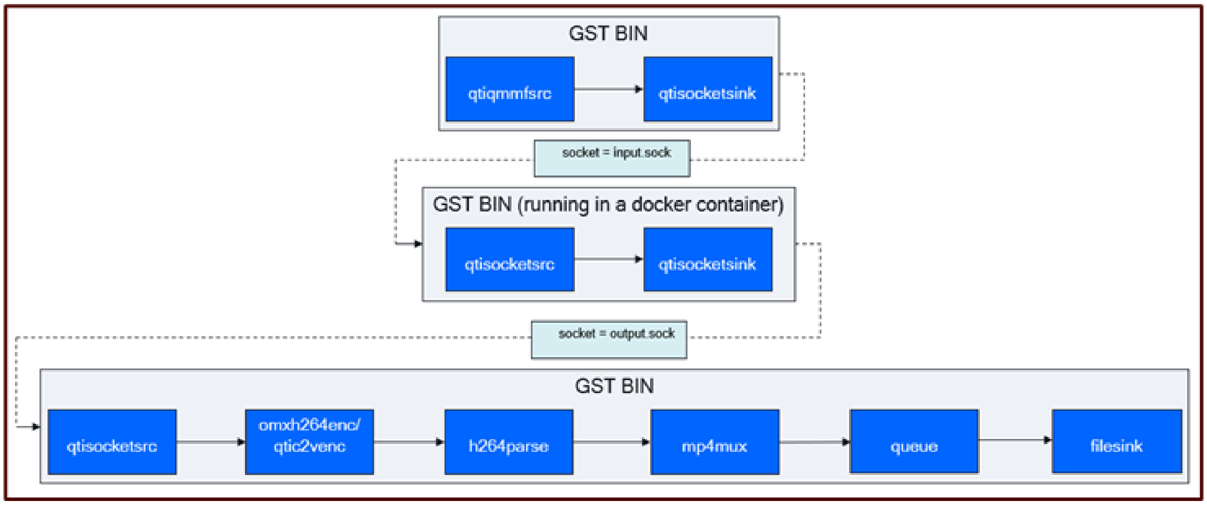


Figure 3-19 Transmitting single H264 encoded camera stream

Command:

```
# Start 3 separate adb shell consoles

# In 1st console:
gst-launch-1.0 -e --gst-debug=2,qtisocketsrc:6 qtisocketsrc socket=/tmp/
docker/output.sock ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080 ! omxh264enc target-bitrate=6000000
periodicity-idr=1 interval-intraframes=29 control-rate=max-bitrate !
h264parse ! mp4mux ! queue ! filesink location=/data/video.mp4

# In 2nd console:
docker run -v /tmp/docker:/tmp/docker -v /usr/lib/gstreamer-1.0/
libqtisocketsrc.so:/usr/lib/aarch64-linux-gnu/gstreamer-1.0/
libqtisocketsrc.so -v /usr/lib/gstreamer-1.0/libqtisocketsink.so:/usr/lib/
aarch64-linux-gnu/gstreamer-1.0/libqtisocketsink.so -ti gst-image /bin/bash -
c "gst-launch-1.0 -e --gst-debug=4 --gst-
debug=2,qtisocketsink:6,qtisocketsrc:6 qtisocketsrc socket=/tmp/docker/
input.sock ! qtisocketsink socket=/tmp/docker/output.sock"

# In 3rd console:
gst-launch-1.0 -e --gst-debug=2,qtisocketsink:6 qtiqmmfsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1,camera=0 !
qtisocketsink socket=/tmp/docker/input.sock
```

3.10 qtic2venc

The qtic2venc GStreamer element uses Codec2 API for encoding a video stream.

This plug-in uses Codec2 video encoder component to provide hardware accelerated H.264 (MPEG-4 Part 10) encoded bit stream on platforms that support it and is based on GstVideoEncoder base class.

Inheritance chain: GObject → GstObject → GstElement → GstVideoEncoder → GstC2_VENCEncoder

qtic2venc pad configuration

Table 3-22 Pad templates for qtic2venc

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	video/x-raw(memory:GBM)	format:	{ (string)NV12, (string)NV21, (string)NV12_10LE32, (string)P010_10LE }
		width:	[1, 2147483647]
		height:	[1, 2147483647]
		framerate:	[0/1, 2147483647/1]
		Max-framerate:	[0/1,2147483647/1]
	video/x-raw	format:	{ (string)NV12, (string)NV21, (string)NV12_10LE32, (string)P010_10LE }

Table 3-22 Pad templates for qtic2venc (cont.)

Pad Name	Capabilities		
		width:	[1, 2147483647]
		height:	[1, 2147483647]
		framerate:	[0/1, 2147483647/1]
		Max-framerate:	[0/1,2147483647/1]
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-h264	stream-format:	{ (string)byte-stream }
		alignment:	{ (string)au }
		framerate:	[0/1,2147483647/1]
	video/x-h265	stream-format:	{ (string)byte-stream }
		alignment:	{ (string)au }
		framerate:	[0/1,2147483647/1]
	video/x-heic	stream-format:	{ (string)byte-stream }
		alignment:	{ (string)au }
	image/x-heic	–	–

qtic2venc element configuration

Table 3-23 Element properties for qtic2venc

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String, Default: "c2_vencencoder0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
qos	■ Handle Quality-of-Service events from downstream ■ flags: readable, writable ■ Boolean. Default: false
control-rate	■ Bitrate control method ■ flags: readable, writable, changeable only in NULL or READY state Enum "GstCodec2VencRateControl" Default: 0, "disable" ■ (0): disable - Disable RC ■ (1): constant - Constant ■ (2): CBR-VFR - Constant bitrate, variable framerate ■ (3): VBR-CFR - Variable bitrate, constant framerate ■ (4): VBR-VFR - Variable bitrate, variable framerate ■ (5): CQ - Constant quality

Table 3-23 Element properties for qtic2venc (cont.)

Property	Description
entropy-mode	▪ Entropy mode for encoding process ▪ flags: readable, writable, changeable only in NULL or READY state ▪ Enum "GstCodec2VencEntropyMode" Default: 0, "none" □ (0): none - None □ (1): cavlc - CAVLC □ (2): cabac - CABAC
idr-interval	▪ IDR frame interval (0 means not explicitly set IDR interval) ▪ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ▪ Unsigned Integer. Range: 0 - 4294967295 Default: 0
iframe-only	▪ I Frame only mode ▪ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ▪ Boolean. Default: false
intra-refresh-mbs	▪ For random modes, it means period of intra refresh. Only support random mode. ▪ flags: readable, writable, changeable only in NULL or READY state ▪ Unsigned Integer. Range: 0 - 4294967295 Default: 0
intra-refresh-mode	▪ Intra refresh mode, only support random mode. Allow IR only for CBR(_CFR/VFR) RC modes ▪ flags:readable, writable, changeable only in NULL or READY state ▪ Enum"GstCodec2VencIntraRefreshMode" Default: 0, "none" □ (0):none - None □ (1):– Arbitrary - arbitrary □ (2) Cyclic - cyclic
intra-refresh-period	▪ Intra Refresh Period. The period of intra refresh. Only support random mode. ▪ flags:readable, writable, changeable only in NULL or READY state ▪ Unsigned Integer. □ Range: 0 - 4294967295 □ Default: 0
loop-filter-mode	▪ Enable or disable the deblocking filter ▪ flags: readable, writable, changeable only in NULL or READY state ▪ Enum "GstCodec2VencLoopFilterMode" Default: 0, "none" □ (0): none - None □ (1): enable - Enable □ (2): disable - Disable □ (3): disable-slice-boundary - Disable-slice-boundary
max-quant-b-frames	▪ Maximum quantization parameter allowed for B-frames ▪ flags: readable, writable, changeable only in NULL or READY state ▪ Unsigned Integer. Range: 0 - 4294967295 Default: 0

Table 3-23 Element properties for qtic2venc (cont.)

Property	Description
max-quant-i-frames	■ Maximum quantization parameter allowed for I-frames ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
max-quant-p-frames	■ Maximum quantization parameter allowed for P-frames ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
min-quant-b-frames	■ Minimum quantization parameter allowed for B-frames ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
min-quant-i-frames	■ Minimum quantization parameter allowed for I-frames ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
min-quant-p-frames	■ Minimum quantization parameter allowed for P-frames ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
num-ltr-frames	■ Number of Long-Term Reference Frames (0xffffffff=component default) ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4294967295
quant-b-frames	■ Quantization parameter for B-frames (0xffffffff=component default) ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4294967295
quant-i-frames	■ Quantization parameter for I-frames (0xffffffff=component default) ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4294967295
quant-p-frames	■ Quantization parameter for P-frames (0xffffffff=component default) ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4294967295
roi-quant-boxes	■ Manually set ROI boxes (e.g., '<<X, Y, W, H, QP>, <X, Y, W, H, QP>>'). The QP values must be in the range of -31 (best quality) to 30 (worst quality) ■ flags: readable, writable, changeable in NULL or READY state ■ GstValueArray of GValues of type "GstValueArray"
roi-quant-meta-value	■ Set specific QP value, different then the default value of (-15), for a GstVideoRegionOfInterestMeta type (e.g., 'roi-meta-qp,person=-20,cup=10,dog=-5;'). The QP values must be in the range of -31 (best quality) to 30 (worst quality) ■ flags: readable, writable, changeable only in NULL or READY state ■ Boxed pointer of type "GstStructure"

Table 3-23 Element properties for qtic2venc (cont.)

Property	Description
roi-quant-mode	■ Enable/Disable Adjustment of the quantization parameter according to ROIs set manually via the 'roi-quant-boxes' property and/or arriving as GstVideoRegionOfInterestMeta attached to the buffer ■ flags: readable, writable, changeable only in NULL or READY state ■ Boolean. Default: false
rotate	■ Rotate video image ■ flags: readable, writable, changeable only in NULL or READY state ■ Enum "GstCodec2VencRotate" Default: 0, "none" □ (0): none - No rotation □ (1): 90CW - Rotate 90 degrees clockwise □ (2): 180 - Rotate 180 degrees □ (3): 90CCW - Rotate 90 degrees counter-clockwise
slice-mode	■ Slice mode, support MB and BYTES mode ■ flags: readable, writable, changeable only in NULL or READY state ■ Enum "GstCodec2VencSliceMode" Default: 0, "disable" □ (0): disable - Slice Mode Disable □ (1): MB - Slice Mode MB □ (2): bytes - Slice Mode Bytes
slice-size	■ Slice size, just set when slice mode setting to MB or Bytes ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
b-frames	■ Number of B-frames between two consecutive P-frames (0=component default) ■ flags: readable, writable, changeable only in NULL or READY state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0
target-bitrate	■ Target bitrate in bits per second (0 means not explicitly set bitrate) ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Unsigned Integer. Range: 0 - 4294967295 Default: 0

Table 3-24 Element actions

Action	Function
trigger-iframe	void user_function (GstElement* object);
ltr-mark	voiduser_function (GstElement* object);

Usage

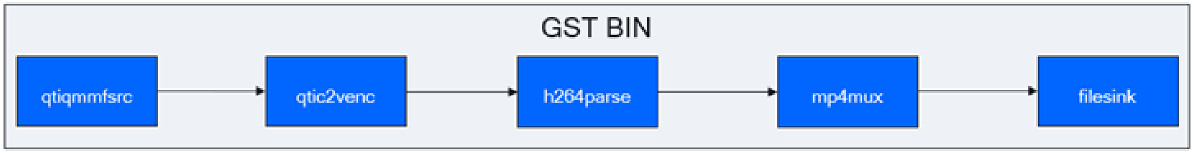


Figure 3-20 Encode and save to filesystem

Command:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! \
  qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/
video.mp4"
```

3.11 qtic2vdec

The qtic2vdec GStreamer element uses Codec2 API for decoding a video stream.

This plug-in uses Codec2 H264 video decoder component to provide hardware-accelerated H.264 (MPEG-4 Part 10) decoding on platforms, which support it and are based on GstVideoDecoder base class.

Inheritance chain: GObject → GstObject → GstElement → GstVideoDecoder → GstC2vdec

qtic2vdec pad configuration

Table 3-25 Pad templates for qtic2vdec

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	video/x-h264	alignment:	au
		stream-format:	byte-stream
	video/x-h265	alignment:	au
		stream-format:	byte-stream
	video/x-vp8	–	–
	video/x-vp9	–	–
	video/mpeg	mpegversion:	2
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-raw(memory:GBM)	format:	NV12
		width:	[32, 8192]
		height:	[32, 8192]
	video/x-raw	format:	NV12
		width:	[32, 8192]
		height:	[32, 8192]

qtic2vdec element configuration

Table 3-26 Element properties for qtic2vdec

Property	Description
name	▪ The name of the object ▪ flags: readable, writable ▪ String, Default: "c2vdec0"
parent	▪ The parent of the object ▪ flags: readable, writable ▪ Object of type "GstObject"

Usage

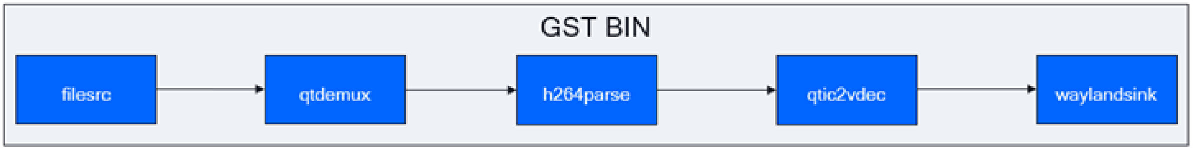


Figure 3-21 Decode and display on screen

Command:

```
gst-launch-1.0 filesrc location=/data/video.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! video/x-raw\(memory:GBM\),compression=ubwc ! queue !
waylandsink fullscreen=true
```

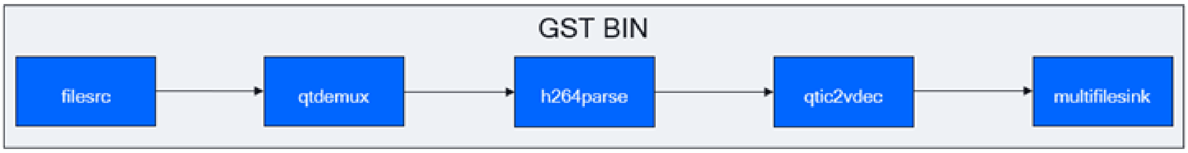


Figure 3-22 Decode and save to filesystem

Command:

```
gst-launch-1.0 filesrc location=/data/video.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! video/x-raw\(memory:GBM\) ! queue ! multifilesink max-
files=5 location=/data/frame_%d.bin
```

3.12 qtijpegenc

The qtijpegenc element utilizes JPEG encoder via camera services to provide hardware-accelerated JPEG encoded stream on platforms that support it and is based on GstVideoEncoder base class. This element can work with camera frames (For example, qtiqmmfsrc) or frames from pre-recorded video file.

Inheritance chain: GObject → GstObject → GstElement → GstVideoEncoder → GstJPEGEncoder

qtijpegenc pad configuration

Table 3-27 Pad templates for qtijpegenc

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	video/x-raw	format:	{ NV12, NV21 }
		width:	[1, 2147483647]
		height:	[1, 2147483647]
		framerate:	[0/1, 2147483647/1]
	video/x-raw(ANY)	format:	{ NV12, NV21 }
		width:	[1, 2147483647]
		height:	[1, 2147483647]
		framerate:	[0/1, 2147483647/1]
SRC template: 'src' *Availability:* Always *Direction:* source	image/jpeg	width:	[1, 65535]
		height:	[1, 65535]
		framerate:	[0/1, 2147483647/1]

qtijpegenc element configuration

Table 3-28 Element properties for qtijpegenc

Property	Description
name	▪ The name of the object ▪ flags: readable, writable ▪ String. Default: "jpegencoder0"
parent	▪ The parent of the object ▪ flags: readable, writable ▪ Object of type "GstObject"
qos	▪ Handle Quality-of-Service events from downstream ▪ flags: readable, writable ▪ Boolean. Default: false
quality	▪ Quality of encoding ▪ flags: readable, writable ▪ Integer. Range: 0 - 100 Default: 85
orientation	▪ Orientation of Jpeg encoder ▪ flags: readable, writable Enum "GstJpegEncodeRotation" Default: 0, "0" ▪ (0): 0 - Orientation 0 degrees ▪ (1): 90 - Orientation 90 degrees

Table 3-28 Element properties for qtijpegenc (cont.)

Property	Description
	■ (2): 180 - Orientation 180 degrees ■ (3): 270 - Orientation 270 degrees
max-req	■ Max request number allowed to submit to JPEGEncoder ■ flags: readable, writable ■ Integer. Range: 0 - 10 Default: 0

Usage

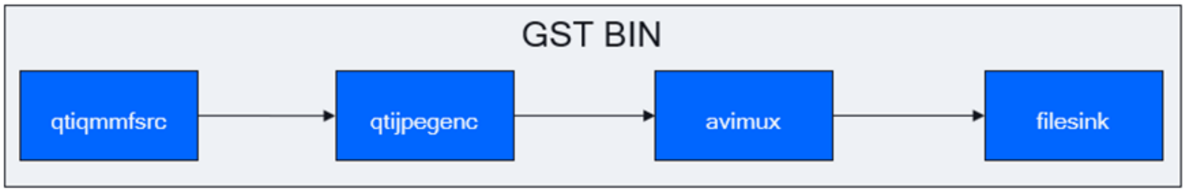


Figure 3-23 Encode YUV420 (NV12) streams and save JPEG video stream in AVI container

Command for 1080p resolution:

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\(memory:GBM\), format=NV12,
width=1920, height=1080, framerate=30/1 ! queue ! qtijpegenc ! image/jpeg !
queue ! avimux ! queue ! filesink location=/data/mjpeg_1080.avi
```

Command for 4K resolution:

```
gst-launch-1.0 qtiqmmfsrc ! video/x-raw\(memory:GBM\), format=NV12,
width=3840, height=2160, framerate=30/1 ! queue ! qtijpegenc ! image/jpeg !
queue ! avimux ! queue ! filesink location=/data/mjpeg_4k.avi
```

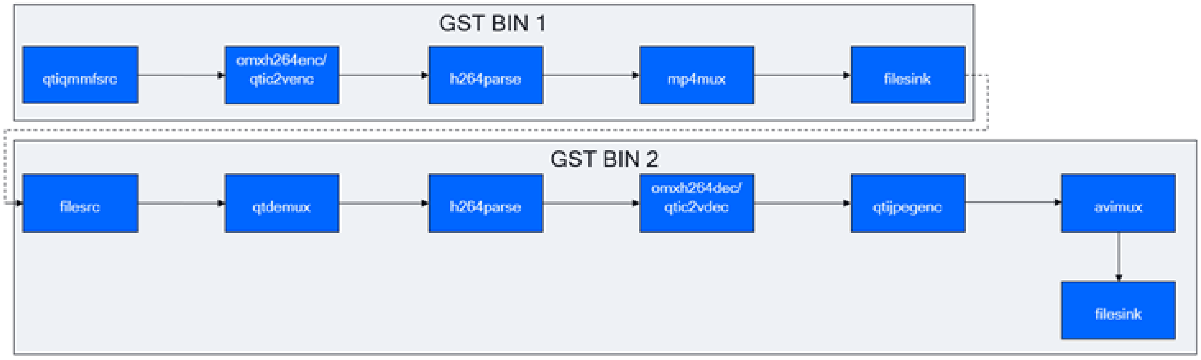


Figure 3-24 Offline reencode in JPEG video stream

Command to record H264 video stream:

```
gst-launch-1.0 -e qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=3840,height=2160,framerate=30/1 ! queue ! qtic2venc !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux_4k.mp4"
```

Command for offline reencode in JPEG:

```
gst-launch-1.0 filesrc location=/data/mux_4k.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! qtijpegenc ! image/jpeg ! avimux ! filesink location=/
data/mjpeg_vid.avi
```

3.13 pulsesink

The pulsesink GStreamer element is a plug-in that interacts with the underlying PulseAudio sound server to play the PCM samples.

The PulseAudio Server interacts with the underlying hardware to provide the playback capability and is based on GstAudioBaseSink base class.

Inheritance chain: GObject → GstObject → GstElement → GstBaseSink → GstAudioBaseSink → GstPulseSink

pulsesink pad configuration

Table 3-29 Pad templates for pulsesink

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	audio/x-raw	format:	{ (string)S16LE, (string)S16BE, (string)F32LE, (string)F32BE, (string)S32LE, (string)S32BE, (string)S24LE, (string)S24BE, (string)S24_32LE, (string)S24_32BE, (string)U8 }
		layout:	interleaved
		rate:	[1, 22579200]
		channels:	[1, 32]
	audio/x-alaw	rate:	[1, 22579200]
		channels:	[1, 32]
	audio/x-mulaw	rate:	[1, 22579200]
		channels:	[1, 32]
	audio/x-ac3	framed:	true
	audio/x-eac3	framed:	true
	audio/x-dts	framed:	true
		block-size:	{ (int)512, (int)1024, (int)2048 }
	audio/mpeg	mpegversion:	1
		mpegaudioversion:	[1, 3]
		parsed:	true
	audio/mpeg	mpegversion:	{ (int)2, (int)4 }
		framed:	true
		stream-format:	adts

pulsesink element configuration

Table 3-30 Element properties for pulsesink

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "pulsesink0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
sync	■ Sync on the clock ■ flags: readable, writable ■ Boolean. Default: true
max-lateness	■ Maximum number of nanoseconds that a buffer can be late before it is dropped (-1 unlimited) ■ flags: readable, writable ■ Integer64. Range: -1 - 9223372036854775807 Default: -1
qos	■ Generate Quality-of-Service events upstream ■ flags: readable, writable ■ Boolean. Default: false
async	■ Go asynchronously to PAUSED ■ flags: readable, writable ■ Boolean. Default: true
ts-offset	■ Timestamp offset in nanoseconds ■ flags: readable, writable ■ Integer64. Range: -9223372036854775808 - 9223372036854775807 Default: 0
enable-last-sample	■ Enable the last-sample property ■ flags: readable, writable ■ Boolean. Default: false
last-sample	■ The last sample received in the sink ■ flags: readable ■ Boxed pointer of type "GstSample"
blocksize	■ Size in bytes to pull per buffer (0 = default) ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4096
render-delay	■ Additional render delay of the sink in nanoseconds ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
throttle-time	■ The time to keep between rendered buffers (0 = disabled) ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0

Table 3-30 Element properties for pulsesink (cont.)

Property	Description
max-bitrate	■ The maximum bits per second to render (0 = disabled) ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
buffer-time	■ Size of audio buffer in microseconds, this is the minimum latency that the sink reports ■ flags: readable, writable ■ Integer64. Range: 1 - 9223372036854775807 Default: 200000
latency-time	■ The minimum amount of data to write in each iteration in microseconds ■ flags: readable, writable ■ Integer64. Range: 1 - 9223372036854775807 Default: 10000
provide-clock	■ Provide a clock to be used as the global pipeline clock ■ flags: readable, writable ■ Boolean. Default: true
slave-method	■ Algorithm used to match the rate of the masterclock ■ flags: readable, writable Enum "GstAudioBaseSinkSlaveMethod" Default: 1, "skew" ■ (0): resample - GST_AUDIO_BASE_SINK_SLAVE_RESAMPLE ■ (1): skew - GST_AUDIO_BASE_SINK_SLAVE_SKEW ■ (2): none - GST_AUDIO_BASE_SINK_SLAVE_NONE ■ (3): custom - GST_AUDIO_BASE_SINK_SLAVE_CUSTOM
can-activate-pull	■ Allow pull-based scheduling ■ flags: readable, writable ■ Boolean. Default: false
alignment-threshold	■ Timestamp alignment threshold in nanoseconds ■ flags: readable, writable ■ Unsigned Integer64. Range: 1 - 18446744073709551614 Default: 40000000
drift-tolerance	■ Tolerance for clock drift in microseconds ■ flags: readable, writable ■ Integer64. Range: 1 - 9223372036854775807 Default: 40000
discont-wait	■ Window of time in nanoseconds to wait before creating a discontinuity ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551614 Default: 1000000000
server	■ The PulseAudio server to connect to ■ flags: readable, writable ■ String. Default: null
device	■ The PulseAudio sink device to connect to ■ flags: readable, writable ■ String. Default: null
current-device	■ The current PulseAudio sink device ■ flags: readable ■ String. Default: ""

Table 3-30 Element properties for pulsesink (cont.)

Property	Description
device-name	■ Human-readable name of the sound device ■ flags: readable ■ String. Default: null
volume	■ Linear volume of this stream, 1.0=100% ■ flags: readable, writable ■ Double. Range: 0 - 10 Default: 1
mute	■ Mute state of this stream ■ flags: readable, writable ■ Boolean. Default: false
client-name	■ The PulseAudio client name to use ■ flags: readable, writable, changeable only in NULL or READY state ■ String. Default: "gst-inspect-1.0"
stream-properties	■ list of pulseaudio stream properties ■ flags: readable, writable ■ Boxed pointer of type "GstStructure"
stream-flags	■ Stream flags to use (not all default flags can be overridden) ■ flags: writable ■ Flags "GstPulseStreamFlags" Default: 0x0000200b, "Try to adjust the latency of the sink/ source based on the requested buffer metrics and adjust buffer metrics accordingly+Timing update requests are issued periodically automatically+Interpolate the latency for this stream +Create the stream corked" □ (0x00000000): Flag to pass when no specific options are needed - GST_PULSE_STREAM_NOFLAGS □ (0x00000001): Create the stream corked - GST_PULSE_STREAM_START_CORKED □ (0x00000002): Interpolate the latency for this stream - GST_PULSE_STREAM_INTERPOLATE_TIMING □ (0x00000004): Don't force the time to increase monotonically - GST_PULSE_STREAM_NOT_MONOTONIC □ (0x00000008): Timing update requests are issued periodically automatically - GST_PULSE_STREAM_AUTO_TIMING_UPDATE □ (0x00000010): Don't remap channels by their name, instead map them simply by their index - GST_PULSE_STREAM_NO_REMAP_CHANNELS □ (0x00000020): When remapping channels by name, don't upmix or downmix them to related channels - GST_PULSE_STREAM_NO_REMIX_CHANNELS □ (0x00000040): Use the sample format of the sink/device this stream is being connected to - GST_PULSE_STREAM_FIX_FORMAT □ (0x00000080): Use the sample rate of the sink - GST_PULSE_STREAM_FIX_RATE □ (0x00000100): Use the number of channels and the channel map of the sink - GST_PULSE_STREAM_FIX_CHANNELS □ (0x00000200): Don't allow moving of this stream to another sink/device - GST_PULSE_STREAM_DONT_MOVE □ (0x00000400): Allow dynamic changing of the sampling rate during playback - GST_PULSE_STREAM_VARIABLE_RATE □ (0x00000800): Find peaks instead of resampling - GST_PULSE_STREAM_PEAK_DETECT □ (0x00001000): Create in muted state - GST_PULSE_STREAM_START_MUTED

Table 3-30 Element properties for pulsesink (cont.)

Property	Description
	□ (0x00002000): Try to adjust the latency of the sink/source based on the requested buffer metrics and adjust buffer metrics accordingly - GST_PULSE_STREAM_ADJUST_LATENCY □ (0x00004000): Enable compatibility mode for legacy clients that rely on a classic hardware device fragment-style playback model - GST_PULSE_STREAM_EARLY_REQUESTS □ (0x00008000): If set this stream won't be considered when it is checked whether the device this stream is connected to should auto-suspend - GST_PULSE_STREAM_DONT_INHIBIT_AUTO_SUSPEND □ (0x00010000): Create in unmuted state - GST_PULSE_STREAM_START_UNMUTED □ (0x00020000): If the sink/source this stream is connected to is suspended during the creation of this stream, cause it to fail - GST_PULSE_STREAM_FAIL_ON_SUSPEND □ (0x00040000): If a volume is passed when this stream is created, creation of this stream, cause it to fail - GST_PULSE_STREAM_RELATIVE_VOLUME □ (0x00080000): Used to tag content that will be rendered by passthrough sinks - GST_PULSE_STREAM_PASSTHROUGH Write only
prebuf	■ Prebuf to be set with sink-input ■ flags: writable ■ Integer. Range: -1 - 2147483647 Default: 0 Write only

Usage

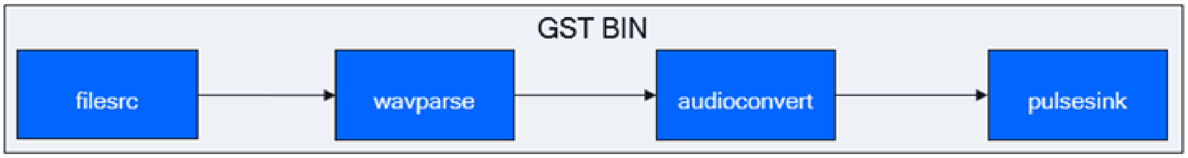


Figure 3-25 Audio playback

Command:

```
gst-launch-1.0 filesrc location=/data/Audio.wav ! wavparse ! audioconvert !
pulsesink volume=10
```

3.14 pulsedirectsink

The pulsedirectsink GStreamer element is a plug-in that interacts with the underlying PulseAudio sound server to play the encoded bit stream via tunneling in the audio digital signal processor (ADSP).

ADSP decodes and sends the decoded samples to the playback device directly.

The PulseAudio Server interacts with the underlying hardware to provide the playback capability and is based on GstBaseSink base class.

Inheritance chain: GObject → GstObject → GstElement → GstBaseSink → GstPulseDirectSink

pulsedirectsink pad configuration

Table 3-31 Pad templates for pulsedirectsink

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	audio/x-raw	format:	{ (string)S16LE, (string)S16BE, (string)F32LE, (string)F32BE, (string)S32LE, (string)S32BE, (string)S24LE, (string)S24BE, (string)S24_32LE, (string)S24_32BE, (string)U8 }
		layout:	interleaved
		rate:	[1, 2147483647]
		channels:	[1, 32]
	audio/x-alaw	rate:	[1, 2147483647]
		channels:	[1, 32]
	audio/x-mulaw	rate:	[1, 2147483647]
		channels:	[1, 32]
	audio/mpeg	mpegversion:	1
		mpegaudioversion:	[1, 3]
		parsed:	true
	audio/mpeg	mpegversion:	{ (int)2, (int)4 }
		framed:	true
		stream-format:	{ (string)adts, (string)raw }
	audio/x-dsd	rate:	[1, 2147483647]
		channels:	[1, 2147483647]

pulsedirectsink element configuration

Table 3-32 Element properties for pulsedirectsink

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "pulsedirectsink0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
sync	■ Sync on the clock ■ flags: readable, writable ■ Boolean. Default: true
max-lateness	■ Maximum number of nanoseconds that a buffer can be late before it is dropped (-1 unlimited) ■ flags: readable, writable ■ Integer64. Range: -1 - 9223372036854775807 Default: -1

Table 3-32 Element properties for pulsedirectsink (cont.)

Property	Description
qos	■ Generate Quality-of-Service events upstream ■ flags: readable, writable ■ Boolean. Default: false
async	■ Go asynchronously to PAUSED ■ flags: readable, writable ■ Boolean. Default: true
ts-offset	■ Timestamp offset in nanoseconds ■ flags: readable, writable ■ Integer64. Range: -9223372036854775808 - 9223372036854775807 Default: 0
enable-last-sample	■ Enable the last-sample property ■ flags: readable, writable ■ Boolean. Default: true
last-sample	■ The last sample received in the sink ■ flags: readable ■ Boxed pointer of type "GstSample"
blocksize	■ Size in bytes to pull per buffer (0 = default) ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4096
render-delay	■ Additional render delay of the sink in nanoseconds ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
throttle-time	■ The time to keep between rendered buffers (0 = disabled) ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
max-bitrate	■ The maximum bits per second to render (0 = disabled) ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
server	■ The PulseAudio server to connect to ■ flags: readable, writable ■ String. Default: null
device	■ The PulseAudio sink device to connect to ■ flags: readable, writable ■ String. Default: null
timestamp	■ Provide buffers with timestamp ■ flags: readable, writable ■ Boolean. Default: false
tlength	■ The target buffer level (total latency) to request (in bytes) ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4294967295

Table 3-32 Element properties for pulsedirectsink (cont.)

Property	Description
minreq	▪ The minmum amount of data that server will request (in bytes) ▪ flags: readable, writable ▪ Unsigned Integer. Range: 0 - 4294967295 Default: 4294967295
maxlength	▪ Maximum stream buffer size that the server should hold (in bytes) ▪ flags: readable, writable ▪ Unsigned Integer. Range: 0 - 4294967295 Default: 4294967295
prebuf	▪ Minimum amount of data required for playback to start (in bytes) ▪ flags: readable, writable ▪ Unsigned Integer. Range: 0 - 4294967295 Default: 4294967295
provide-clock	▪ Provide a clock that can be used as the pipeline clock ▪ flags: readable, writable ▪ Boolean. Default: true
stream-properties	▪ list of pulseaudio stream properties ▪ flags: readable, writable ▪ Boxed pointer of type "GstStructure"

Usage

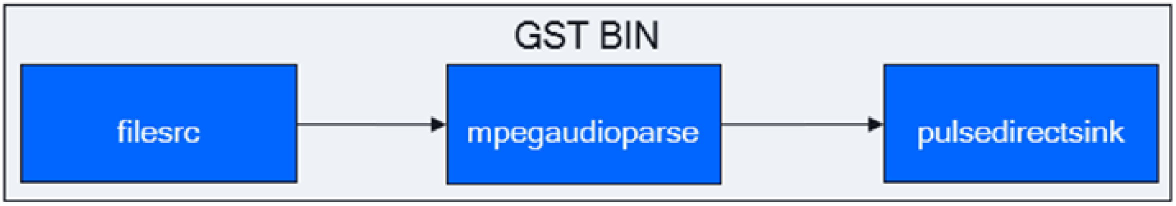


Figure 3-26 Audio playback (MP3)

Command:

```
gst-launch-1.0 filesrc location=/data/AudioMp3.mp3 ! mpegaudioparse !
pulsedirectsink
```

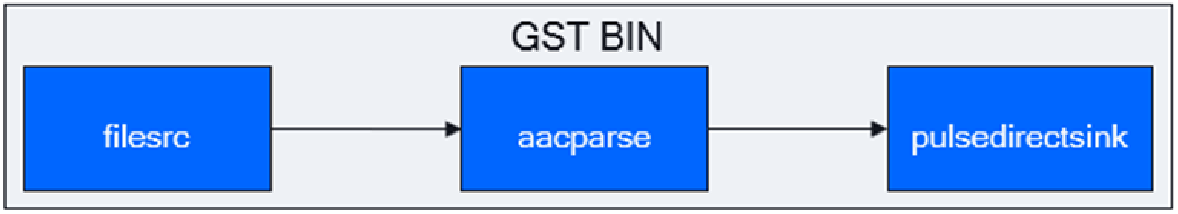


Figure 3-27 Audio playback (AAC)

Command:

```
gst-launch-1.0 filesrc location=/data/AudioAAC.aac ! aacparse !
pulsedirectsink
```

3.15 waylandsink

The waylandsink element uses GstVideoSink GStreamer class. It is based on Wayland Weston compositor, which creates its own window and renders the incoming video frames in the window.

It maps the Weston client APIs and states to the appropriate GStreamer APIs and states. QTI has extended the waylandsink implementation and added support for window positioning, proprietary GBM-based buffer backend for zero-copy and XDG shell backend.

Inheritance chain: GObject → GstObject → GstElement → GstBaseSink → GstVideoSink → GstWaylandSink

waylandsink pad configuration

Table 3-33 Pad templates for waylandsink

Pad Name	Capabilities		
SINK template: 'sink' Availability:Always Direction: sink	video/x-raw	format:	{ (string)BGRx, (string)BGRA, (string)RGBx, (string)xBGR, (string)xRGB, (string)RGBA, (string)ABGR, (string)ARGB, (string)RGB, (string)BGR, (string)RGB16, (string)BGR16, (string)YUY2, (string)YVYU, (string)UYVY, (string)AYUV, (string)NV12, (string)NV21, (string)NV16, (string)YUV9, (string)YVU9, (string)Y41B, (string)I420, (string)YV12, (string)Y42B, (string)v308 }
		width:	[1, 2147483647]
		height:	[1, 2147483647]
		framerate:	[0/1, 2147483647/1]
	video/x-raw(memory:DMABuf)	format:	{ (string)BGRx, (string)BGRA, (string)RGBx, (string)xBGR, (string)xRGB, (string)RGBA, (string)ABGR, (string)ARGB, (string)RGB, (string)BGR, (string)RGB16, (string)BGR16, (string)YUY2, (string)YVYU, (string)UYVY, (string)AYUV, (string)NV12, (string)NV21, (string)NV16, (string)YUV9, (string)YVU9, (string)Y41B, (string)I420, (string)YV12, (string)Y42B, (string)v308 }
		width:	[1, 2147483647]
		height:	[1, 2147483647]
		framerate:	[0/1, 2147483647/1]
	video/x-raw(memory:GBM)	format:	{ (string)BGRx, (string)BGRA, (string)RGBx, (string)xBGR, (string)xRGB, (string)RGBA, (string)ABGR, (string)ARGB, (string)RGB, (string)BGR, (string)RGB16, (string)BGR16, (string)YUY2, (string)YVYU, (string)UYVY, (string)AYUV, (string)NV12, (string)NV21, (string)NV16, (string)YUV9, (string)YVU9, (string)Y41B, (string)I420, (string)YV12, (string)Y42B, (string)v308 }

Table 3-33 Pad templates for waylandsink (cont.)

Pad Name	Capabilities		
		width:	[1, 2147483647]
		height:	[1, 2147483647]
		framerate:	[0/1, 2147483647/1]

waylandsink element configuration

Table 3-34 Element properties for waylandsink

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "waylandsink0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
sync	■ Sync on the clock ■ flags: readable, writable ■ Boolean. Default: true
max-lateness	■ Maximum number of nanoseconds that a buffer can be late before it is dropped (-1 unlimited) ■ flags: readable, writable ■ Integer64. Range: -1 - 9223372036854775807 Default: 20000000
qos	■ Generate Quality-of-Service events upstream ■ flags: readable, writable ■ Boolean. Default: true
async	■ Go asynchronously to PAUSED ■ flags: readable, writable ■ Boolean. Default: true
ts-offset	■ Timestamp offset in nanoseconds ■ flags: readable, writable ■ Integer64. Range: -9223372036854775808 - 9223372036854775807 Default: 0
enable-last-sample	■ Enable the last-sample property ■ flags: readable, writable ■ Boolean. Default: true
last-sample	■ The last sample received in the sink ■ flags: readable ■ Boxed pointer of type "GstSample"
blocksize	■ Size in bytes to pull per buffer (0 = default) ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 4294967295 Default: 4096

Table 3-34 Element properties for waylandsink (cont.)

Property	Description
render-delay	■ Additional render delay of the sink in nanoseconds ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
throttle-time	■ The time to keep between rendered buffers (0 = disabled) ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
max-bitrate	■ The maximum bits per second to render (0 = disabled) ■ flags: readable, writable ■ Unsigned Integer64. Range: 0 - 18446744073709551615 Default: 0
show-preroll-frame	■ Whether to render video frames during preroll ■ flags: readable, writable ■ Boolean. Default: true
display	■ Wayland display name to connect to, if not supplied via the GstContext ■ flags: readable, writable ■ String. Default: null
xdg-shell	■ Whether to use the XDG shell protocol for the display ■ flags: readable, writable ■ Boolean. Default: true
fullscreen	■ Whether the surface should be made fullscreen ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Boolean. Default: false
x	■ X position for the content ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Unsigned Integer. Range: 0 - 1920 Default: 0
y	■ Y position for the content ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Unsigned Integer. Range: 0 - 1920 Default: 0
width	■ Destination Width for the content ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Unsigned Integer. Range: 0 - 1920 Default: 640
height	■ Destination Height for the content ■ flags: readable, writable, changeable in NULL, READY, PAUSED or PLAYING state ■ Unsigned Integer. Range: 0 - 1080 Default: 480

Usage

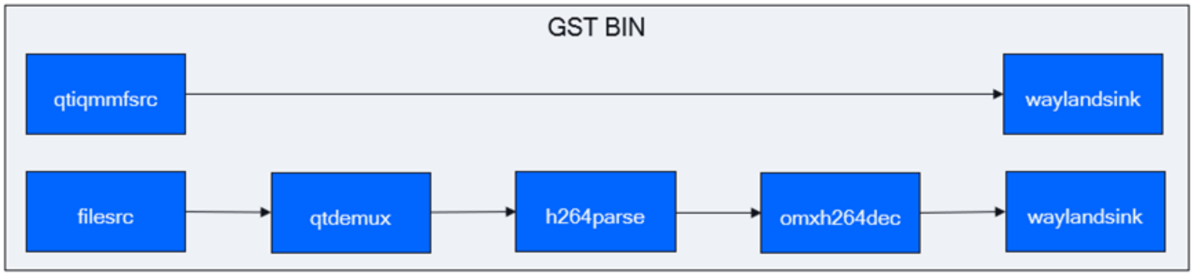


Figure 3-28 Single camera stream and single H264 file stream Picture-in-Picture mode

Command:

```
gst-launch-1.0 -e --gst-debug=2 \
filesrc location=/data/Street_Side_720p_180s_30FPS.MOV ! qtdemux ! queue max-
size-bytes=0 max-size-time=0 ! h264parse !  qtic2vdec ! queue ! waylandsink
fullscreen=true sync=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! waylandsink
x=1280 y=720 width=640 height=360 sync=false
```

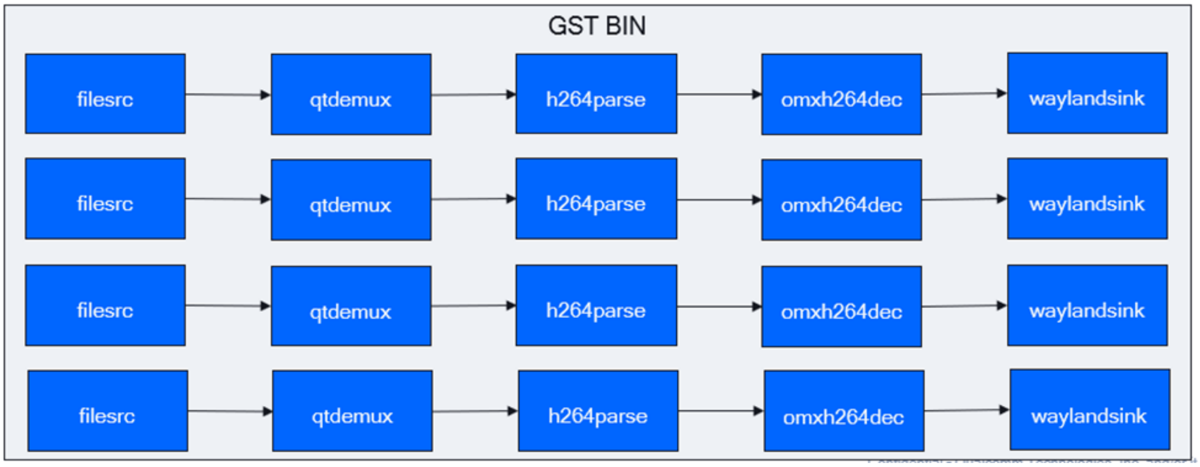


Figure 3-29 Multiple H264 file streams with custom destinations

Command:

```
gst-launch-1.0 -e --gst-debug=2 \
filesrc location=/data/Street_Side_720p_180s_30FPS.MOV ! qtdemux ! queue !
h264parse !  qtic2vdec ! queue ! waylandsink x=0 y=0 width=960 height=540
sync=true \
filesrc location=/data/Carview_720p_180s_30FPS.MOV ! qtdemux ! queue !
h264parse !  qtic2vdec ! queue ! waylandsink x=960 y=0 width=960 height=540
sync=true \
filesrc location=/data/Driving_720p_180s_30FPS.MOV ! qtdemux ! queue !
h264parse !  qtic2vdec ! queue ! waylandsink x=0 y=720 width=960 height=540
sync=true \
filesrc location=/data/Street_Bridge_720p_180s_30FPS.MOV ! qtdemux ! queue !
```

```
h264parse !  qtic2vdec ! queue ! waylandsink x=960 y=720 width=960 height=540
sync=true
```

3.16 qticvoptclflow

The qticvoptclflow (optical workflow) plug-in provides motion vector/estimation of objects in scene by leveraging CV hardware block to calculate the movement predictions.

NOTE This plug-in will be enabled in a future release.

The input data source for this plug-in can be either live camera or offline stream. By default, both the motion vectors and their accompanying statistics are transmitted downstream. The statistics can be disabled via the stats property.

The motion vector prediction, in the form of metadata attached to buffer (this is done with the help of qtimetamux), can be processed by the qtioverlay plug-in to provide arrow visualization on screen or be send to another layer for further analysis.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstCvOptclFlow

qticvoptclflow pad configuration

Table 3-35 Pad templates and qticvoptclflow

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	video/x-raw	format:	{ GRAY8, NV12 }
		width:	[1, 2147483647]
		height:	[1, 2147483647]
		framerate:	[0/1, 2147483647/1]
	video/x-raw	format:	{ GRAY8, NV12 }
		width:	[1, 2147483647]
		height:	[1, 2147483647]
		framerate:	[0/1, 2147483647/1]
SRC template: 'src' *Availability:* Always *Direction:* source	cvp/x-optical-flow	–	–

qticvoptclflow element configuration

Table 3-36 Element properties of qticvoptclflow

Property	Description
name	▪ The name of the object ▪ flags: readable, writable ▪ String, Default: "cvoptclflow0"
parent	▪ The parent of the object ▪ flags: readable, writable ▪ Object of type "GstObject"

Table 3-36 Element properties of qticvoptclflow (cont.)

Property	Description
stats	■ Enable statistics for additional motion vector info ■ flags: readable, writable ■ Boolean. Default: true
qos	■ Handle Quality-of-Service events ■ flags: readable, writable ■ Boolean. Default: false

Usage

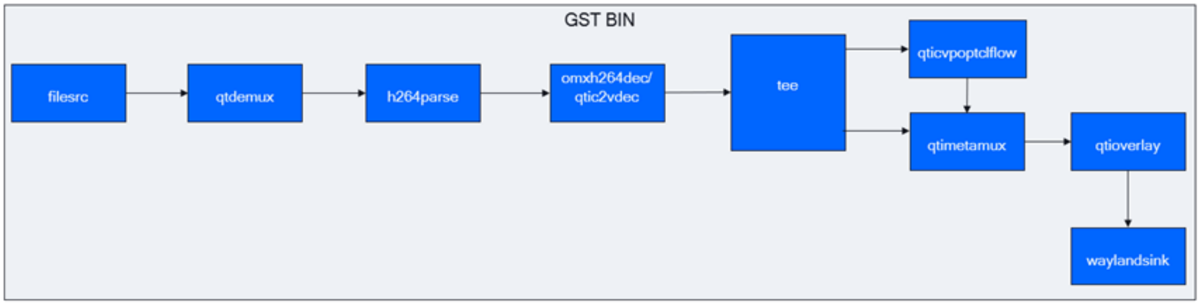


Figure 3-30 Optical flow on H264 file source using display sink

Command:

```
gst-launch-1.0 -e filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux !
queue ! h264parse ! qtic2vdec ! queue ! tee name=split split. ! queue !
qtimetamux name=metamux ! queue ! qtioverlay ! queue ! waylandsink
fullscreen=true sync=false split. ! queue ! qticvoptclflow ! metamux.
```

3.17 qticvpimgpyramid

The qticvpimgpyramid (image pyramid) plug-in provides multiple down-scaled images from single input image by leveraging CVP hardware block.

The input data source can be either be live camera or offline stream. By default, all the 20 (including the base image) level/scales are generated, but the image generation is can be controlled by num-octaves and num-scales property.

The output scaled images are grayscale and can be fed to composer for rendering, ML chain as input, or dumped to file.

Inheritance chain: GObject → GstObject → GstElement → GstCvpImgPyramid

qticvpimgpyramid pad configuration

Table 3-37 Pad templates for qticvpimgpyramid

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always	video/x-raw	format:	{ GRAY8, NV12 }
		width:	[1, 32767]

Table 3-37 Pad templates for qticvpimgpyramid (cont.)

Pad Name	Capabilities		
Direction: sink		height:	[1, 32767]
		framerate:	[0/1, 2147483647/1]
	video/x-raw (memory:GBM)	format:	{ GRAY8, NV12 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 2147483647/1]
SRC template: 'src_%u' *Availability:* On request *Direction:* source	video/x-raw	format:	{ GRAY8, NV12 }
		width:	[1, 32767]
		height:	[1, 32767]
		framerate:	[0/1, 2147483647/1]

qticvpimgpyramid element configuration

Table 3-38 Element properties for qticvpimgpyramid

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String, Default: "cvpimgpyramid0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
num-octaves	■ Number of layers in the pyramid where the resolution is halved ■ flags: readable, writable ■ Unsigned Integer. Range: 1 - 5 Default: 5
num-scales	■ Number of intermediate layers in the pyramid between two octaves ■ flags: readable, writable ■ Unsigned Integer. Range: 1 - 4 Default: 4
octave-sharpness	■ Array of coefficients. The size of this array is equal to the number ofoctaves (n_octaves). Format is <c1, c2, c3, cn>.The value range per octave [0-4], with default 3 ■ flags: readable, writable ■ GstValueArray of GValues of type "guint"

Usage

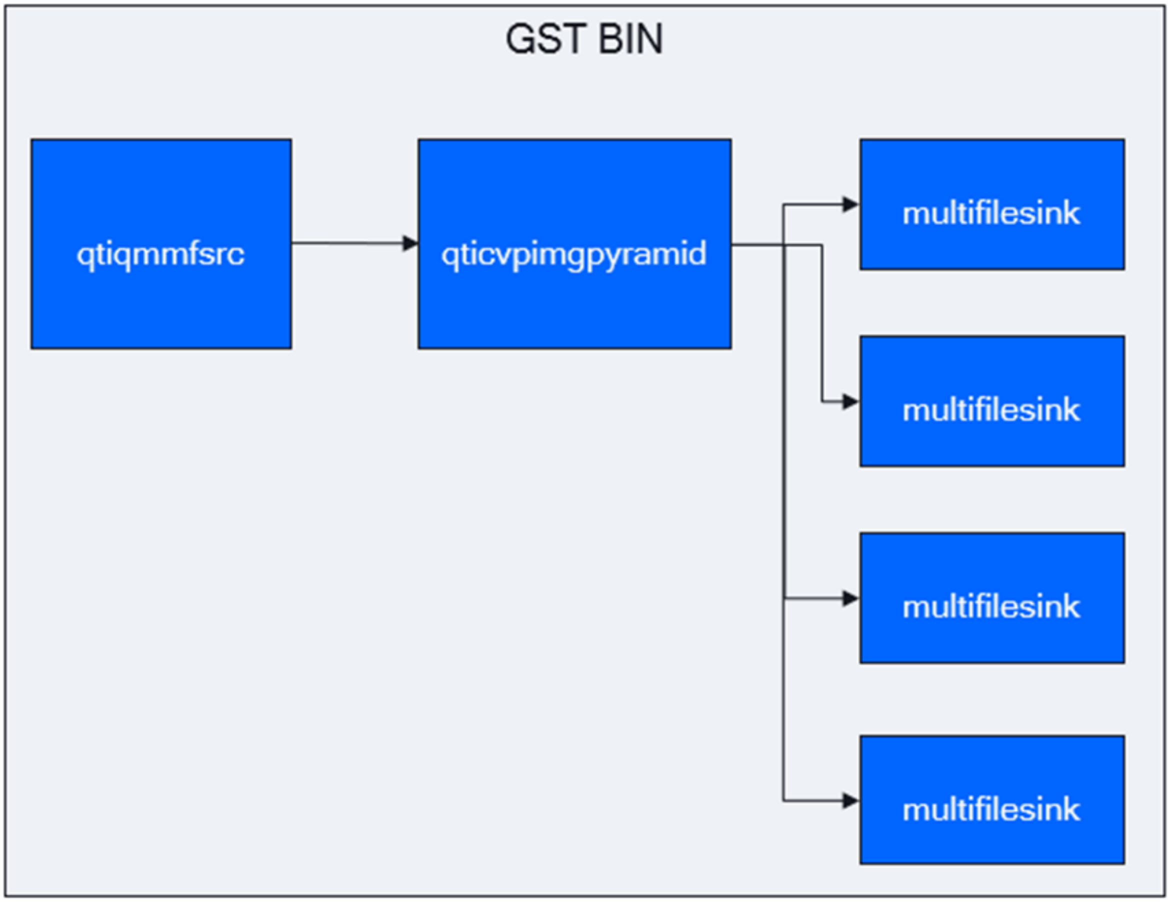


Figure 3-31 Four downscale outputs from a live camera input, and write output to a file

Use CVP image pyramid to generate 4*1/2 downscale buffers from a live camera input:

```
gst-launch-1.0 -f qtiqmmfsrc name=camsrc ! \
video/x-raw\(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 !
qticvpimgpyramid name=pyscal num-octaves=5 num-scales=1 pyscal.src_1 !
multifilesink location=/data/frame_1_%d.yuv max-files=1 \
pyscal.src_2 ! multifilesink location=/data/frame_2_%d.yuv max-files=1 \
pyscal.src_3 ! multifilesink location=/data/frame_3_%d.yuv max-files=1 \
pyscal.src_4 ! multifilesink location=/data/frame_4_%d.yuv max-files=1
```

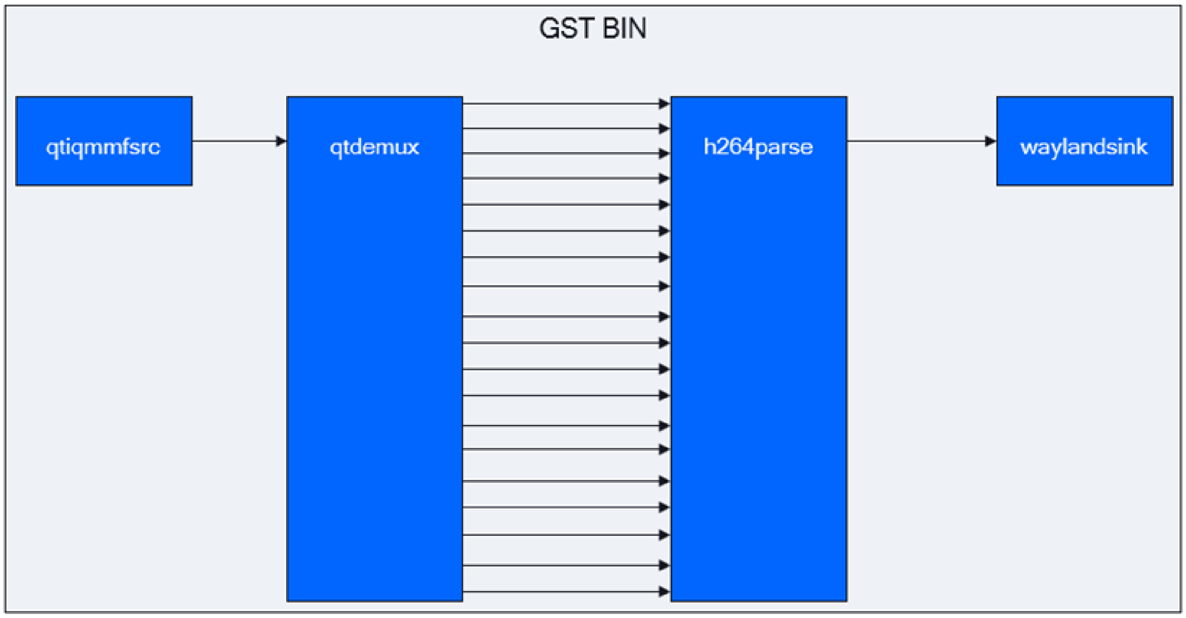


Figure 3-32 Downscale outputs from a live camera input

Use CVP image pyramid to generate 19 downscale buffers from a live camera input:

```
export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 qtiqmmfsrc
name=camsrc ! \
video/x-raw\(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 !
qticvpimgpyramid name=pyscal num-octaves=5 num-scales=4 pyscal.src_1 ! \
qtivcomposer name=mixer sink_0::position="<0, 0>" sink_0::dimensions="<808,
454>" sink_1::position="<0, 454>" sink_1::dimensions="<679, 382>" \
sink_2::position="<808, 0>" sink_2::dimensions="<571, 321>"
sink_3::position="<808, 321>" sink_3::dimensions="<480, 270>"  \
sink_4::position="<808, 591>" sink_4::dimensions="<404, 227>"
sink_5::position="<808, 818>" sink_5::dimensions="<340, 191>"  \
sink_6::position="<1379, 0>" sink_6::dimensions="<286, 161>"
sink_7::position="<1379, 161>" sink_7::dimensions="<240, 135>"  \
sink_8::position="<1379, 296>" sink_8::dimensions="<202, 114>"
sink_9::position="<1379, 410>" sink_9::dimensions="<170, 96>" \
sink_10::position="<1379, 506>" sink_10::dimensions="<143, 81>"
sink_11::position="<1379, 587>" sink_11::dimensions="<120, 68>" \
sink_12::position="<1379, 655>" sink_12::dimensions="<101, 57>"
sink_13::position="<1379, 712>" sink_13::dimensions="<85, 48>"  \
sink_14::position="<1379, 760>" sink_14::dimensions="<72, 40>"
sink_15::position="<1379, 800>" sink_15::dimensions="<60, 34>"  \
sink_16::position="<1379, 834>" sink_16::dimensions="<51, 28>"
sink_17::position="<1379, 862>" sink_17::dimensions="<43, 24>"  \
sink_18::position="<1379, 886>" sink_18::dimensions="<36, 20>" ! queue !
waylandsink enable-last-sample=false async=false fullscreen=true \
pyscal.src_2 ! mixer. pyscal.src_3 ! mixer. pyscal.src_4 ! mixer.
pyscal.src_5 ! mixer. pyscal.src_6 ! mixer. pyscal.src_7 ! mixer. \
pyscal.src_8 ! mixer. pyscal.src_9 ! mixer. pyscal.src_10 ! mixer.
pyscal.src_11 ! mixer. pyscal.src_12 ! mixer. pyscal.src_13 ! mixer. \
pyscal.src_14 ! mixer. pyscal.src_15 ! mixer. pyscal.src_16 ! mixer.
pyscal.src_17 ! mixer. pyscal.src_18 ! mixer. pyscal.src_19 ! mixer.
```

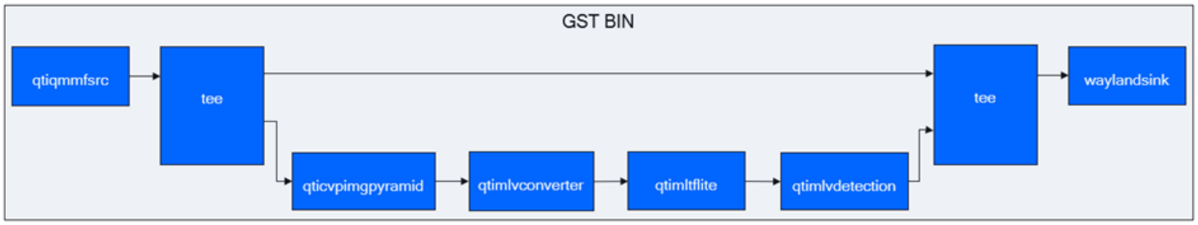


Figure 3-33 Downscale image for ML detection

Use CVP image pyramid to generate a downscale buffer from a live camera input for ML detection:

```
export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 -e --gst-debug=2
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
waylandsink sync=false fullscreen=true split. ! queue ! qticvpimgpyramid
name=pyscal num-octaves=5 num-scales=1 pyscal.src_2 ! qtimlvconverter !
queue ! qtimltflite delegate=gpu model=/data/yolov5m-320x320-int8.tflite !
queue ! qtimlvdetection threshold=75.0 results=10 module=yolov5m labels=/data/
yolov5m.labels ! text/x-raw ! queue ! metamux.
```

3.18 qtimlvconverter

The qtimlvconverter element transforms incoming video buffers into neural-network tensors while performing necessary format conversion and resizing in the process. To achieve these operations the plug-in leverages GPU hardware and ION/DMA allocated buffers.

For floating point tensors, the mean and sigma properties are required in order to properly transform the 8-bit unsigned integer images coming from the video source using the formula: (channel - mean) x sigma, where channel is the value for either Red, Green, Blue or Alpha channels of the color converted input.

For example, an RGB image with values R=255, G=127, B=0 and properties mean="<128.0, 156.0, 124.0, 58.0>" and sigma="<0.75, 0.34, 0.02, 0.07>" would have the following channel transformations: R=(255 - 128.0) x 0.75, G=(127 - 156.0) x 0.34, B=(0 - 124.0) x 0.02.

Arrangement of the image pixels in the output tensor can be specified with the subpixel-layout property.

Under the hood, the plug-in uses either Qualcomm® Adreno™ GPU C2D or QTI IB2C library for all conversion operations. This library is wrapped inside the custom GstC2dVideoConverter or GstGlesVideoConverter abstraction layer respectively with APIs to create, configure and process the incoming and outgoing buffers.

The output buffers are allocated by a custom buffer pool class called GstMLBufferPool, which can allocate ION buffers through IOCTL commands to the kernel.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstMLVideoConverter

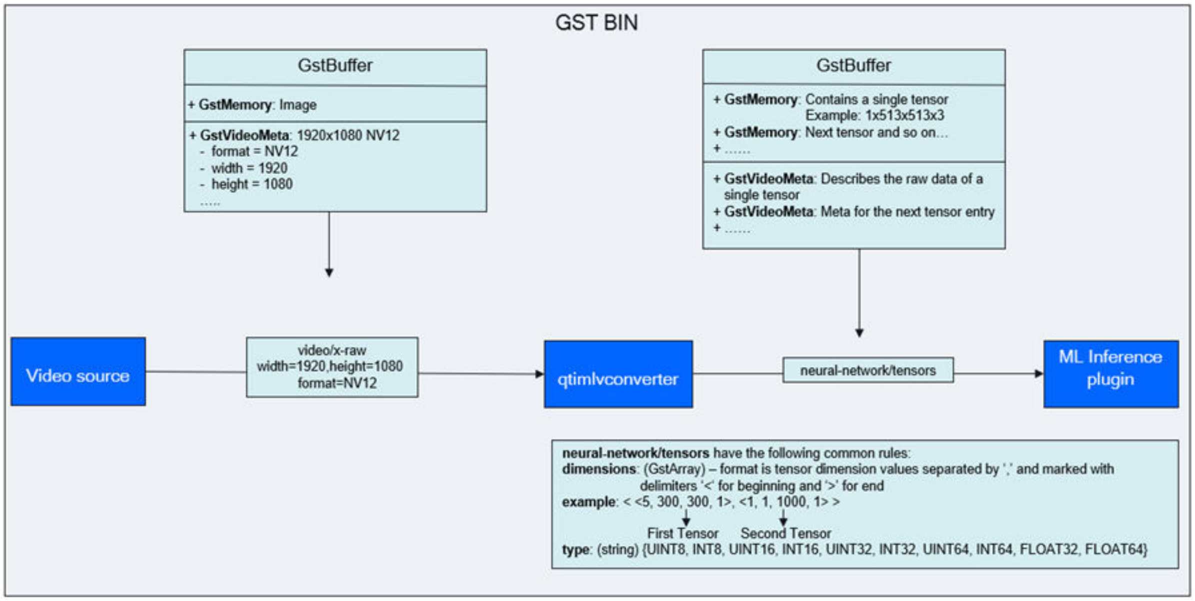

GST BIN
GstBuffer
+ GstMemory: Image
+ GstVideoMeta: 1920x1080 NV12
- format = NV12
- width = 1920
- height = 1080
.....
GstBuffer
+ GstMemory: Contains a single tensor Example: 1x513x513x3
+ GstMemory: Next tensor and so on…
+ ……
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ……
Video source
video/x-raw width=1920,height=1080 format=NV12
qtimlvconverter
neural-network/tensors
ML Inference plugin
neural-network/tensors have the following common rules:
dimensions: (GstArray) – format is tensor dimension values separated by ',' and marked with delimiters '<' for beginning and '>' for end
example: < <5, 300, 300, 1>, <1, 1, 1000, 1> >
First Tensor
Second Tensor
type: (string) {UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64}

Figure 3-34 ML processing architecture

qtimlvconverter pad configuration

Table 3-39 Pad templates for qtimlvconverter

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	video/x-raw	format:	{ (string)RGBA, (string)BGRA, (string)ABGR, (string)ARGB, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)BGR, (string)RGB, (string)GRAY8, (string)NV12, (string)NV21, (string)YUY2, (string)UYVY }
	video/x-raw(memory:GBM)	format:	{ (string)RGBA, (string)BGRA, (string)ABGR, (string)ARGB, (string)RGBx, (string)BGRx, (string)xRGB, (string)xBGR, (string)BGR, (string)RGB, (string)GRAY8, (string)NV12, (string)NV21, (string)YUY2, (string)UYVY }
SRC template: 'src' *Availability:* Always *Direction:* source	neural-network/tensors	type:	{ (string)UINT8, (string)INT32, (string)FLOAT16, (string)FLOAT32 }

qtimlvconverter element configuration

Table 3-40 Element properties for qtimlvconverter

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "mlvideoconverter0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
qos	■ Handle Quality-of-Service events ■ flags: readable, writable ■ Boolean. Default: false
subpixel-layout	■ Arrangement of the image pixels in the output tensor ■ flags: readable, writable Enum "GstMLVideoPixelLayout" Default: 0, "regular" ■ (0): regular - Regular subpixel layout e.g., RGB, RGBA, RGBx, etc. ■ (1): reverse - Reverse subpixel layout e.g., BGR, BGRA, BGRx, etc.
mean	■ Channels mean subtraction values for FLOAT tensors ('<R, G, B>', '<R, G, B, A>', '<G>') ■ flags: readable, writable ■ GstValueArray of GValues of type "gdouble"
sigma	■ Channel divisor values for FLOAT tensors ('<R, G, B>', '<R, G, B, A>', '<G>') ■ flags: readable, writable ■ GstValueArray of GValues of type "gdouble"
engine	■ Choose fastcv or gles as the backend of qtimlvconverter ■ Value could be “fcv” or “gles” ■ For Non-GPU device, default value is “fcv”, for GPU device, default value is “gles ”

Usage

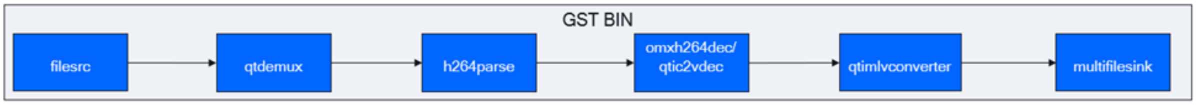


Figure 3-35 Single H264 file stream with manually set UINT8 ML GstCaps and output tensors saved in separate files

Command:

```
gst-launch-1.0 -e --gst-debug=2 filesrc location=/data/
Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue ! h264parse ! qtic2vdec ! queue !
\ qtimlvconverter engine=fcv ! neural-network/
tensors,type=UINT8,dimensions="<<1,300,300,3>>" ! multifilesink location=/
data/tensor_u8_%d.bin
```

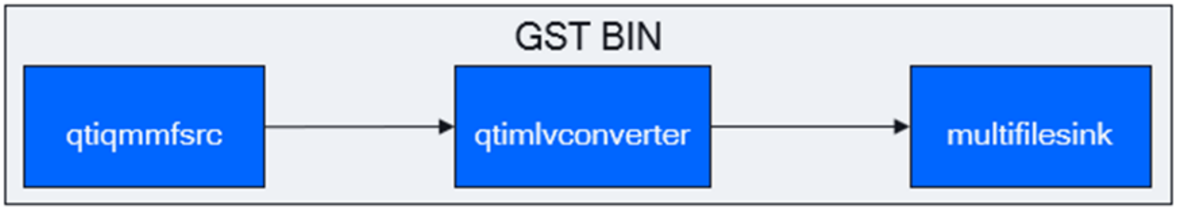


Figure 3-36 Single camera stream with manually set FLOAT32 ML GstCaps and output tensors saved in separate files

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! \
qtimlvconverter engine=gles mean="<128.0, 128.0, 128.0>" sigma="<1.0, 1.0,
1.0>" ! neural- network/tensors,type=FLOAT32,dimensions="<<1,300,300,3>>" !
multifilesink location=/data/tensor_f32_%d.bin
```

3.19 qtimlsnpe

The qtimlsnpe plug-in exposes the Qualcomm® Neural Processing SDK capabilities (load and execute the models). It accepts a tensor prepared by pre-processing elements such as qtimlvconverter and produces output tensor that can be parsed by postprocessing plug-ins such as qtimlvclassification, qtimlvdetection, qtimlvsegmentation, and qtimlvpose.

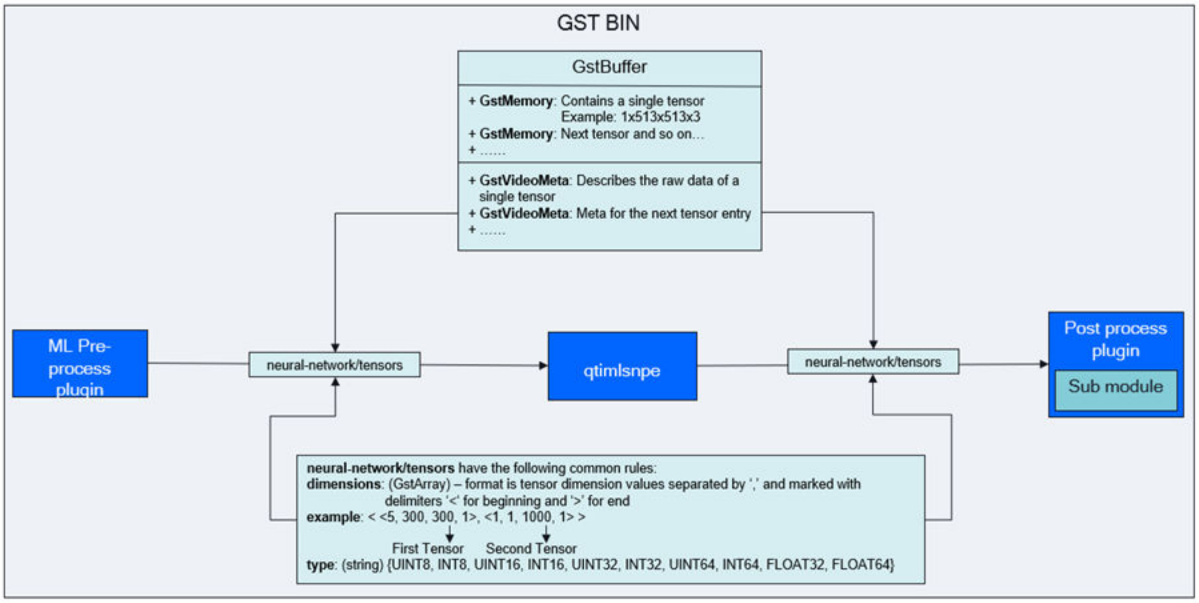
You must specify the model which is going to be used by providing the absolute path to it via the model property. Once loaded the model file will provide the input and output capabilities for the whole plug-in e.g., number of tensors, their dimensions and type.

For some models, setting the additional layers property may be required.

If no delegate is specified, the plug-in will execute the model operations on CPU. The user can select different delegate to use by setting the delegate property. More details on them can be found in the element properties.

Under the hood in addition to the SNPE library the plug-in uses ION/DMA buffers allocated by a custom buffer pool class GstMLBufferPool through IOCTL commands to the kernel.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstMLSnpe



neural-network/tensors have the following common rules:
dimensions: (GstArray) – format is tensor dimension values separated by ',' and marked with delimiters '<' for beginning and '>' for end
example: < <5, 300, 300, 1>, <1, 1, 1000, 1> >
type: (string) {UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64}

Figure 3-37 Tensor mode qtimlsnpe architecture

qtimlsnpe pad configuration

Table 3-41 Pad templates for qtimlsnpe

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	neural-network/tensors	type:	{ (string)UINT8, (string)INT32, (string)FLOAT32 }
SRC template: 'src' *Availability:* Always *Direction:* source	neural-network/tensors	type:	{ (string)UINT8, (string)INT32, (string)FLOAT32 }

qtimlsnpe element configuration

Table 3-42 Element properties of qtimlsnpe

Property	Description
name	▪ The name of the object ▪ flags: readable, writable ▪ String. Default: "mlsnpe0"
parent	▪ The parent of the object ▪ flags: readable, writable ▪ Object of type "GstObject"
model	▪ Model filename ▪ flags: readable, writable ▪ String. Default: null

Table 3-42 Element properties of qtimlsnpe (cont.)

Property	Description
delegate	■ Delegate the graph execution to another executor ■ flags: readable, writable Enum "GstMLSnpeDelegate" Default: 0, "none" ■ (0): none - No delegate, CPU is used for all operations ■ (1): dsp - Run the processing on the Hexagon DSP ■ (2): gpu - Run the processing on the Adreno GPU ■ (3): aip - Run the processing on Snapdragon AIX + HVX
layers	■ List of output layers. Should be set if model has more than one output ■ flags: readable, writable ■ GstValueArray of GValues of type "gchararray"

Usage

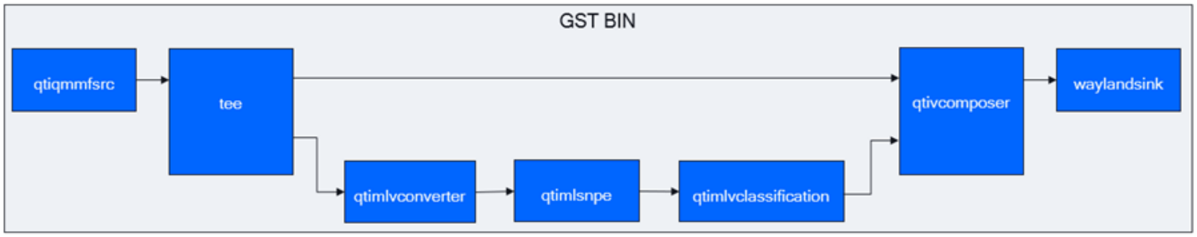


Figure 3-38 Single camera stream with image classification using video composer and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::position="<50, 50>" sink_1::dimensions="<368, 32>" ! queue !
waylandsink sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=dsp model=/data/
resnet50_enhanced_quantized.dlc ! queue ! qtimlvclassification threshold=51.0
results=1 module=mobilenet labels=/data/resnet50.labels ! video/x-
raw,format=BGRA,width=368,height=32 ! queue ! mixer
```

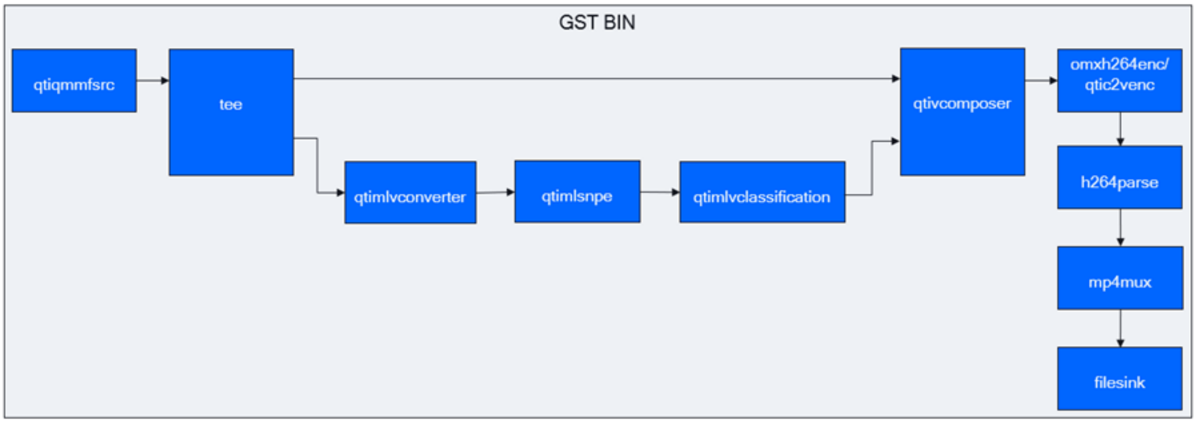


Figure 3-39 Single camera stream with image classification using video composer and saved to file

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::position="<50, 50>" sink_1::dimensions="<368, 32>" ! queue !
qtic2venc ! h264parse ! queue ! mp4mux ! queue ! filesink location=/data/
video.mp4 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=dsp model=/data/
resnet50_enhanced_quantized.dlc ! queue ! qtimlvclassification threshold=51.0
results=1 module=mobilenet labels=/data/resnet50.labels ! video/x-
raw,format=BGRA,width=368,height=32 ! queue ! mixer
```

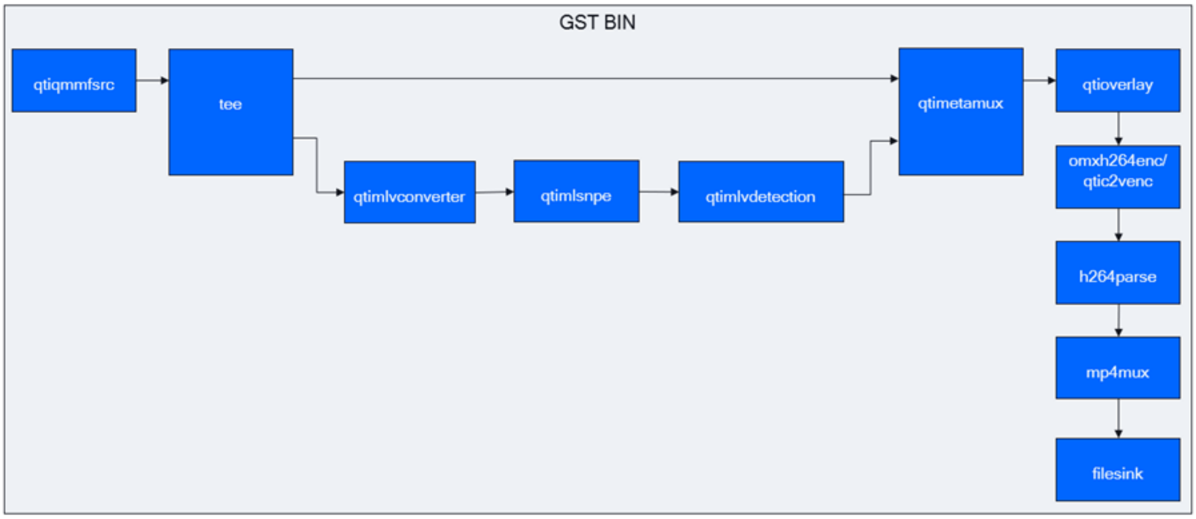


Figure 3-40 Single camera stream with object detection using video overlay and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue !
qtioverlay ! queue ! waylandsink sync=true fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! metamux. \
split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=dsp model=/
home/od/MobileNet_v1_ssd_2017_quantized.dlc layers="<Postprocessor/
BatchMultiClassNonMaxSuppression>" ! queue ! qtimlvdetection threshold=51.0
results=10 module=ssd-mobilenet labels=/home/od/ssd-MobileNet.labels ! text/x-
raw ! queue ! metamux
```

3.20 qtimltflite

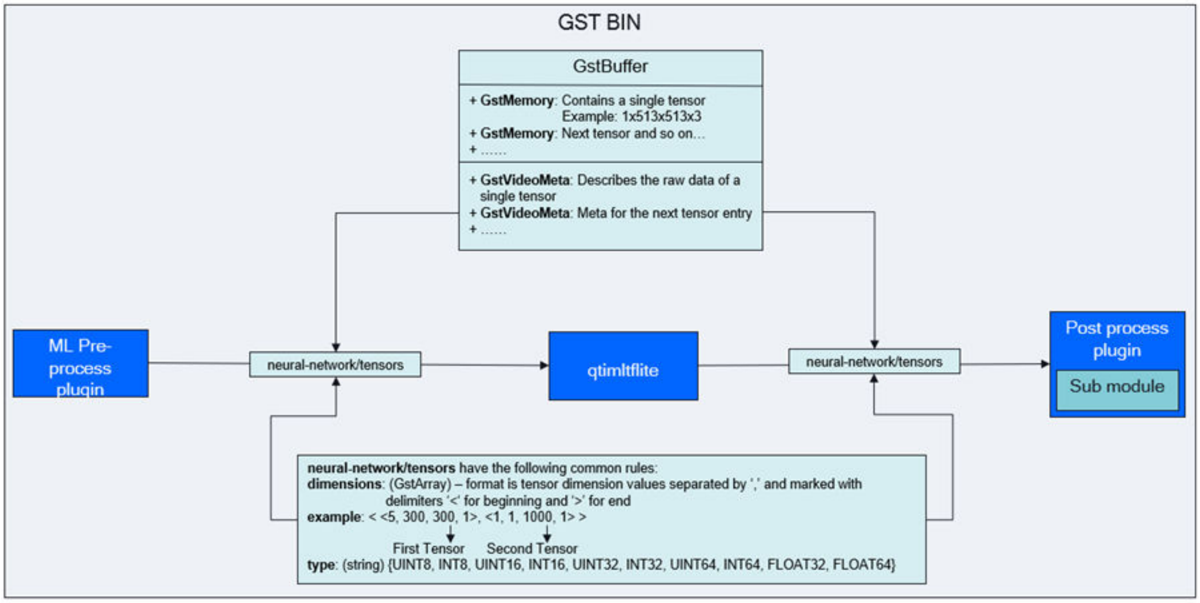
The qtimltflite element exposes the TensorFlow Lite capabilities (load and execute the TensorFlow Lite models) as a GStreamer plug-in. It accepts a tensor prepared by pre-processing elements such as qtimlvconverter and produces output tensor that can be parsed by postprocessing plug-ins such as qtimlvclassification, qtimlvdetection, qtimlvsegmentation, and qtimlvpose.

You must specify the model that is going to be used by providing the absolute path to it via the model property. After it's loaded, the model file provides the input and output capabilities for the whole plug-in. For example, the number of tensors, their dimensions and type.

If no delegate is specified, the plug-in will execute the model operations on CPU. The user can select different delegate to use by setting the delegate property. For more information, see the element properties table.

Under the hood, in addition to the TensorFlow Lite library, the plug-in uses ION/DMA buffers allocated by a custom buffer pool class GstMLBufferPool through IOCTL commands to the kernel.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstMLTFLite


GST BIN
GstBuffer
+ GstMemory: Contains a single tensor
Example: 1x513x513x3
+ GstMemory: Next tensor and so on...
+ ......
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ......
ML Pre-process plugin
neural-network/tensors
qtimltflite
neural-network/tensors
Post process plugin
Sub module
neural-network/tensors have the following common rules:
dimensions: (GstArray) – format is tensor dimension values separated by ',' and marked with delimiters '<' for beginning and '>' for end
example: < <5, 300, 300, 1>, <1, 1, 1000, 1> >
First Tensor
Second Tensor
type: (string) {UINT8, INT8, UINT16, INT16, UINT32, INT32, UINT64, INT64, FLOAT32, FLOAT64}

qtimltflite pad configuration

Table 3-43 Pad templates for qtimltflite

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* Always *Direction:* sink	neural-network/tensors	type:	{ (string)UINT8, (string)INT32, (string)FLOAT32 }
SRC template: 'src' *Availability:* Always *Direction:* source	neural-network/tensors	type:	{ (string)UINT8, (string)INT32, (string)FLOAT32 }

qtimltflite element configuration

Table 3-44 Element properties for qtimltflite

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "mlTFLite0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
qos	■ Handle Quality-of-Service events ■ flags: readable, writable ■ Boolean. Default: false
model	■ Model filename ■ flags: readable, writable ■ String. Default: null
delegate	■ Delegate part or all of graph execution to another executor ■ flags: readable, writable Enum "GstMLTFLiteDelegate" Default: 0, "none" ■ (0): none - No delegate, CPU is used for all operations ■ (1): nnapi-dsp - Run the processing on the DSP through the Android NN API. Unsupported operations will fall back on NPU, GPU or CPU ■ (2): nnapi-gpu - Run the processing on the GPU through the Android NN API. Unsupported operations will fall back on DSP, NPU or CPU ■ (3): nnapi-npu - Run the processing on the NPU through the Android NN API. Unsupported operations will fall back on DSP, GPU, or CPU ■ (4): hexagon - Run the processing directly on the Hexagon DSP ■ (5): gpu - Run the processing directly on the GPU ■ (6): xnnpack - Run inferences using xnnpack cpu runtime
threads	■ Number of threads when running operations on CPU delegate ■ flags: readable, writable ■ Unsigned Integer. Range: 1 - 4 Default: 1

Usage

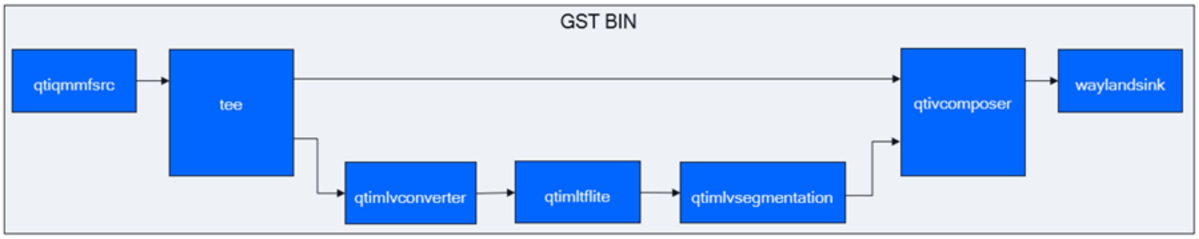


Figure 3-41 Single camera stream with image segmentation using video composer and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::dimensions="<1920,1080>" sink_1::alpha=0.5 ! queue ! waylandsink
sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/dv3_argmax_int32.tflite ! queue ! qtimlvsegmentation module=deeplab-
argmax labels=/data/dv3-argmax.labels ! video/x-raw,width=256,height=144 !
queue ! mixer
```

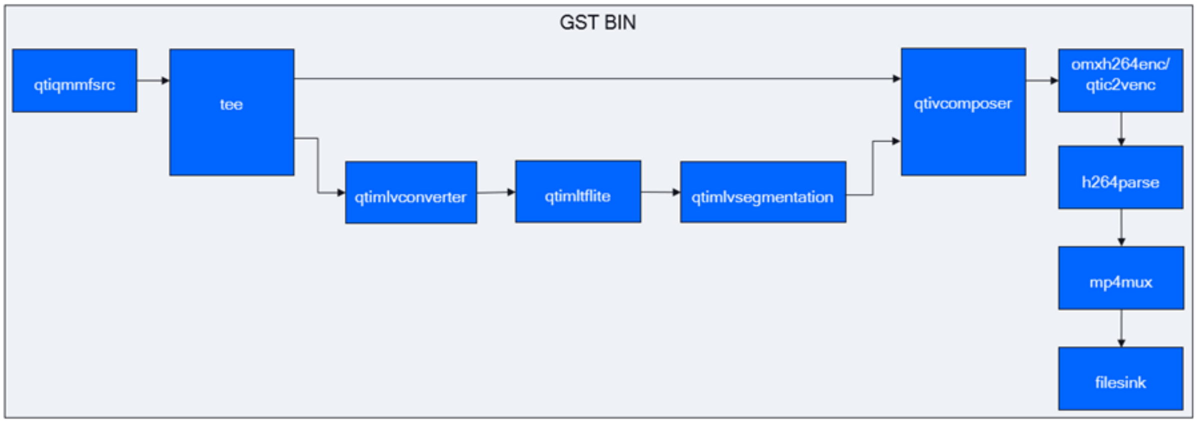


Figure 3-42 Single camera stream with image segmentation using video composer and saved to file

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::dimensions="<1920,1080>" sink_1::alpha=0.5 ! queue ! qtic2venc !
h264parse ! queue ! mp4mux ! queue ! filesink location=/data/video.mp4 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/dv3_argmax_int32.tflite ! queue ! qtimlvsegmentation module=deeplab-
```

```
argmax labels=/data/dv3-argmax.labels ! video/x-raw,width=256,height=144 !
queue ! mixer
```

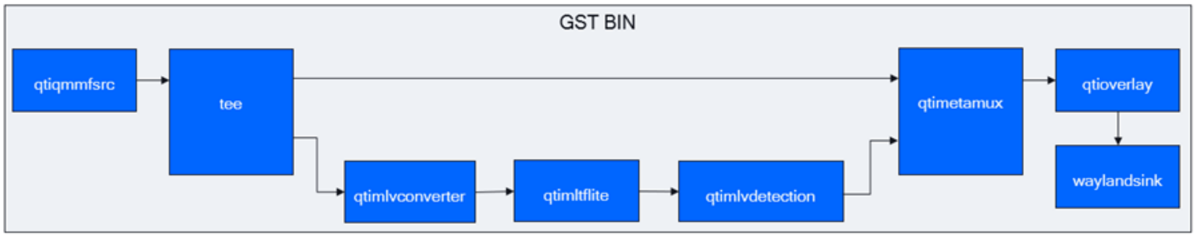


Figure 3-43 Single camera stream with object detection using video overlay and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue !
qtioverlay ! queue ! waylandsink sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! metamux. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection threshold=75.0
results=10 module=yolov5m labels=/data/yolov5m.labels ! text/x-raw ! queue !
metamux
```

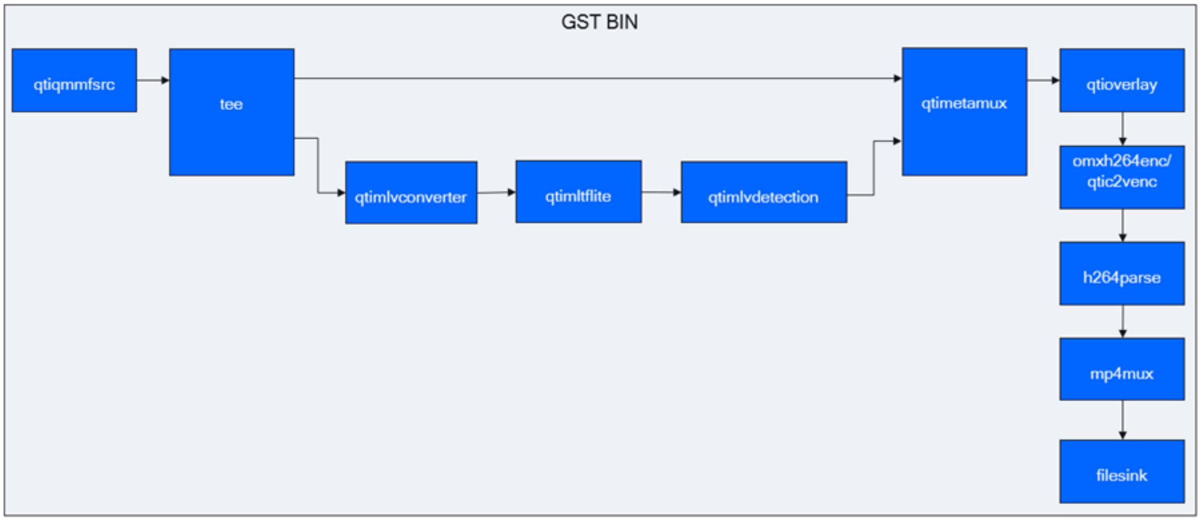


Figure 3-44 Single camera stream with object detection using video overlay and saved to file

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue !
qtioverlay ! queue ! qtic2venc ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! metamux. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
```

```
data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection threshold=75.0
results=10 module=yolov5m labels=/data/yolov5m.labels ! text/x-raw ! queue !
metamux
```

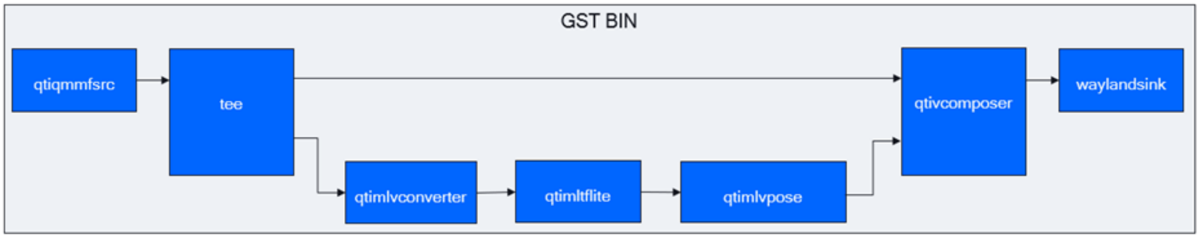


Figure 3-45 Single camera stream with pose estimation using video composer and displayed on screen

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::dimensions="<1920,1080>" ! queue ! waylandsink sync=false
fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/posenet_MobileNet_v1_075_481_641_quant.tflite ! queue ! qtimlvpose
threshold=51.0 results=2 module=posenet labels=/data/posenet.labels ! video/x-
raw,format=BGRA,width=640,height=360 ! queue ! mixer
```

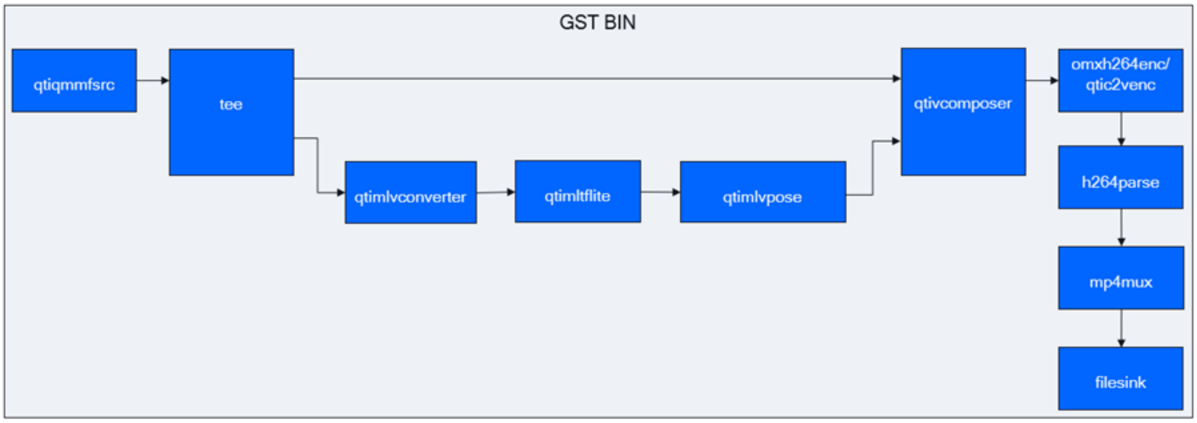


Figure 3-46 Single camera stream with pose estimation using video overlay and saved to file

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue !
qtioverlay ! queue ! qtic2venc ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! metamux. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
```

```
data/posenet_MobileNet_v1_075_481_641_quant.tflite ! queue ! qtimlvpose
threshold=51.0 results=2 module=posenet labels=/data/posenet.labels ! text/x-
raw ! queue ! metamux
```

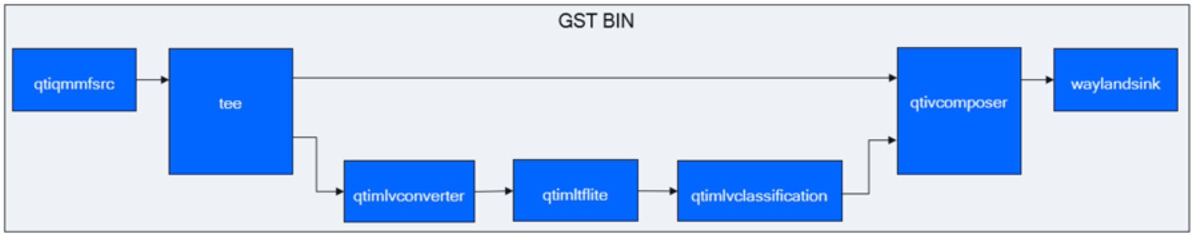


Figure 3-47 Single camera stream with image classification using video composer and displayed on screen

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::position="<50, 50>" sink_1::dimensions="<368, 64>" ! queue !
waylandsink sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/mobilenet_v2_1.0_224_quant.tflite ! queue ! qtimlvclassification
threshold=60.0 results=3 module=mobilenet labels=/data/mobilenet.labels !
video/x-raw,format=BGRA,width=368,height=64 ! queue ! mixer
```

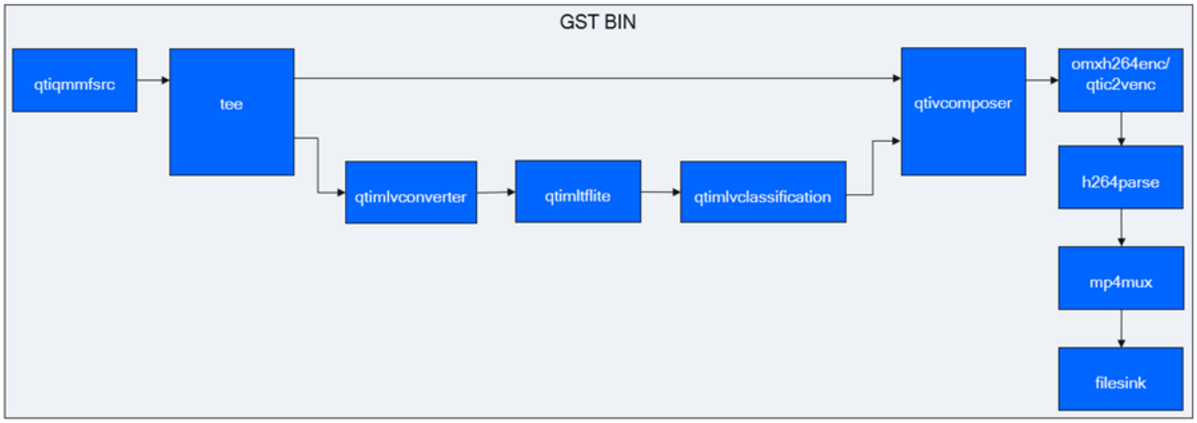


Figure 3-48 Single camera stream with image classification using video composer and saved to file

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::position="<50, 50>" sink_1::dimensions="<368, 64>" ! queue !
qtic2venc! h264parse ! queue ! mp4mux ! queue ! filesink location=/data/
video.mp4 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
```

```
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/mobilenet_v2_1.0_224_quant.tflite ! queue ! qtimlvclassification
threshold=60.0 results=3 module=mobilenet labels=/data/mobilenet.labels !
video/x-raw,format=BGRA,width=368,height=64 ! queue ! mixer
```

3.21 qtimlvdetection

The qtimlvdetection element processes output tensors of an Object Detection model from ML inference plug-in (such as qtimltflite, qtimlsnpe and qtimlaic) into result of predictions.

he processed output is determined by the negotiated GstCaps on the plug-in output. It can be either an image mask (GstCaps: video/x-raw) which for an example can be applied over the original image using qtivcomposer or GStreamer formatted text (GstCaps: text/x-raw) containing the prediction results.

For image overlay mask, the element will leverage the CPU based Cairo 2D graphics library to draw the prediction results in ION/DMA buffers allocated by the custom buffer pool class GstImageBufferPool through IOCTL commands to the kernel. While in the versatile text format the prediction results will be parsed into GStreamer formatted string inside buffers allocated using regular system memory.

The method used for this postprocessing operations is determined by the module and labels properties of the plug-in. The module property specifies which post-process module to run and is populated dynamically at run time with the libraries available in `/usr/lib/gstreamer-1.0/ml/modules/` containing the prefix "`ml-vdetection-`" and the labels property is a customized text file different for each machine learning detection model that user needs to provide for the prediction labels.

Optional properties are available for adjusting the prediction results. Use results to control the number of results displayed and use threshold to set a confidence threshold for prediction, results with confidence below that won't be displayed.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstMLVideoDetection

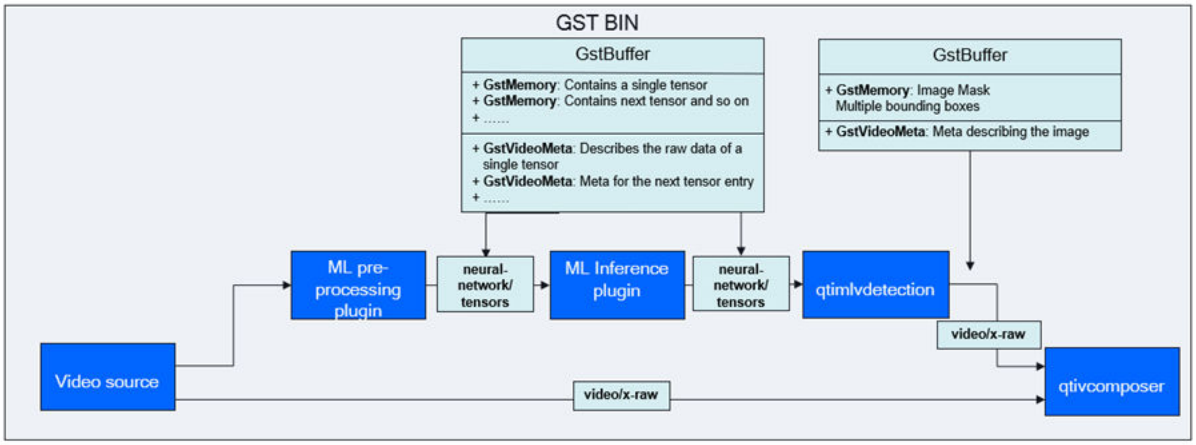

GST BIN
GstBuffer
+ GstMemory: Contains a single tensor
+ GstMemory: Contains next tensor and so on
+ ......
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ......
GstBuffer
+ GstMemory: Image Mask Multiple bounding boxes
+ GstVideoMeta: Meta describing the image
ML pre-processing plugin
neural-network/ tensors
ML Inference plugin
neural-network/ tensors
qtimlvdetection
video/x-raw
Video source
video/x-raw
qtivcomposer

Figure 3-49 Postprocessing for object detection architecture

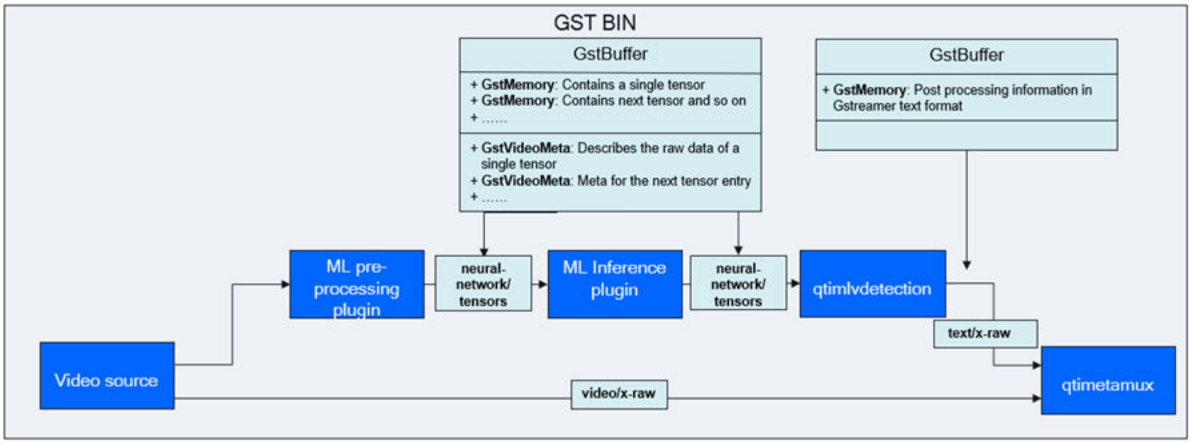

GST BIN
GstBuffer
+ GstMemory: Contains a single tensor
+ GstMemory: Contains next tensor and so on
+ ......
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ......
GstBuffer
+ GstMemory: Post processing information in Gstreamer text format
Video source
ML pre-processing plugin
neural-network/tensors
ML Inference plugin
neural-network/tensors
qtimlvdetection
text/x-raw
video/x-raw
qtimetamux

Figure 3-50 Qtimlvdetection in GStreamer pipeline

qtimlvdetection pad configuration

Table 3-45 Pad templates for qtimlvdetection

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* On request *Direction:* sink	neural-network/tensors	–	–
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-raw	format:	{ (string)BGRA, (string)BGRx, (string)BGR16 }
	video/x-raw(memory:GBM)	format:	{ (string)BGRA, (string)BGRx, (string)BGR16 }
	text/x-raw	format:	{ (string)utf8 }

qtimlvdetection element configuration

Table 3-46 Element properties for qtimlvdetection

Property	Description
name	▪ The name of the object ▪ flags: readable, writable ▪ String. Default: "mlvideodetection0"
parent	▪ The parent of the object ▪ flags: readable, writable ▪ Object of type "GstObject"
qos	▪ Handle Quality-of-Service events ▪ flags: readable, writable ▪ Boolean. Default: false
module	▪ Module name that is going to be used for processing the tensors ▪ flags: readable, writable

Table 3-46 Element properties for qtimlvdetection (cont.)

Property	Description
	Enum "GstMLVideoDetectionModules" (0): none - No module, default invalid mode ■ (1): face-detect - ml-vdetection-face-detect ■ (2): ssd-mobilenet - ml-vdetection-ssd-MobileNet ■ (3): yolov5m - ml-vdetection-yolov5m ■ (4): yolov5s - ml-vdetection-yolov5s
labels	■ Labels filename ■ flags: readable, writable ■ String. Default: null
results	■ Number of results to display ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 10 Default: 5
threshold	■ Confidence threshold in % ■ flags: readable, writable ■ Double. Range: 10.0 - 100.0 Default: 10.0

Usage

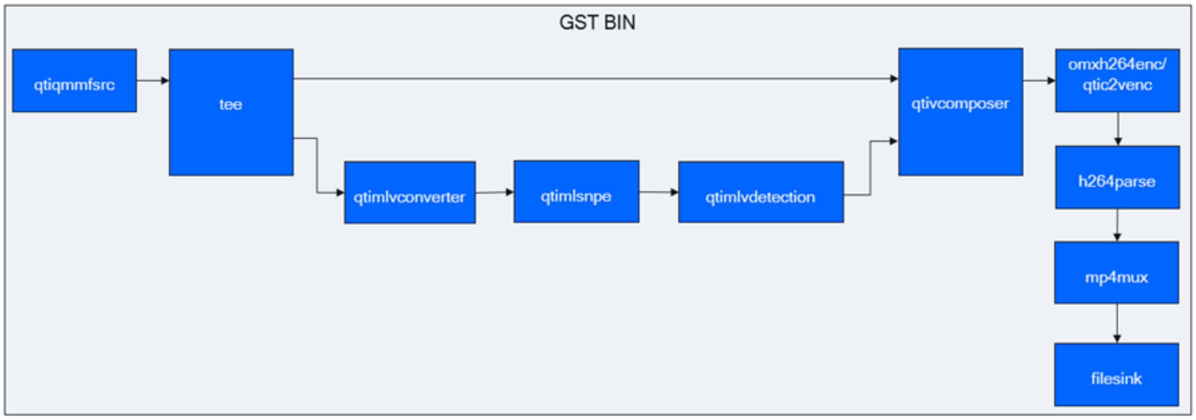


Figure 3-51 Single camera stream with object detection using Video Composer and saved to file

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::dimensions="<1920, 1080>" ! queue ! qtic2venc ! h264parse ! queue !
mp4mux ! queue ! filesink location=/data/video.mp4 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=dsp model=/data/
mobilenet_v1_ssd_2017_quantized.dlc layers="<Postprocessor/
BatchMultiClassNonMaxSuppression>" ! queue ! \
qtimlvclassification threshold=60.0 results=3 module=mobilenet labels=/data/
```

```
mobilenet.labels ! video/x-raw,format=BGRA,width=640,height=360 ! queue !
mixer
```

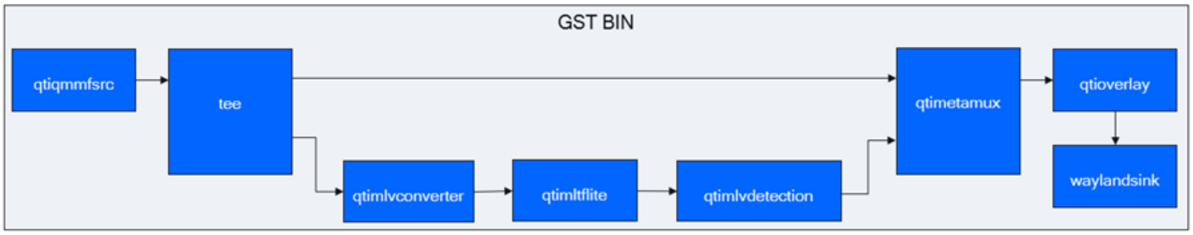


Figure 3-52 High and low-resolution camera streams with object detection using video overlay and displayed on screen

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue ! qtioverlay
text-font-size=24 ! queue ! waylandsink sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! metamux. \
camrsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=640,height=360,framerate=30/1 ! queue !
qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/data/
yolov5m-320x320-int8.tflite ! queue ! \
qtimlvdetection threshold=75.0 results=10 module=yolov5m labels=/data/
yolov5m.labels ! text/x-raw ! queue ! metamux
```

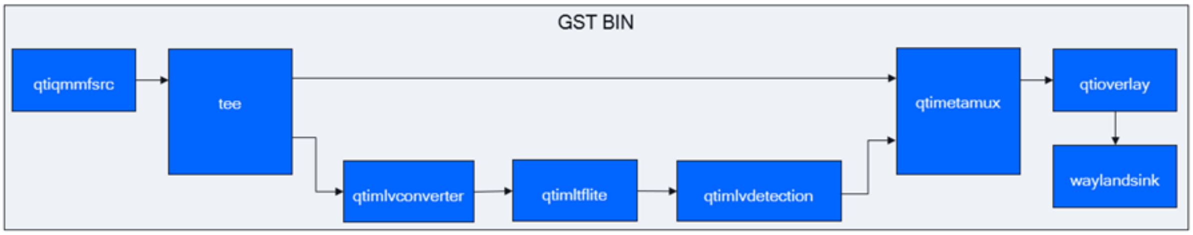


Figure 3-53 Single H264 file stream with object detection using video composer and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::dimensions="<1280, 720>" ! queue ! waylandsink sync=false
fullscreen=true \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse !  qtic2vdec ! queue ! tee name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=dsp model=/data/
mobilenet_v1_ssd_2017_quantized.dlc layers="<Postprocessor/
BatchMultiClassNonMaxSuppression>" ! queue ! \
qtimlvdetection threshold=51.0 results=10 module=ssd-mobilenet labels=/data/
ssd-MobileNet.labels ! video/x-raw,format=BGRA,width=640,height=360 ! queue !
mixer
```

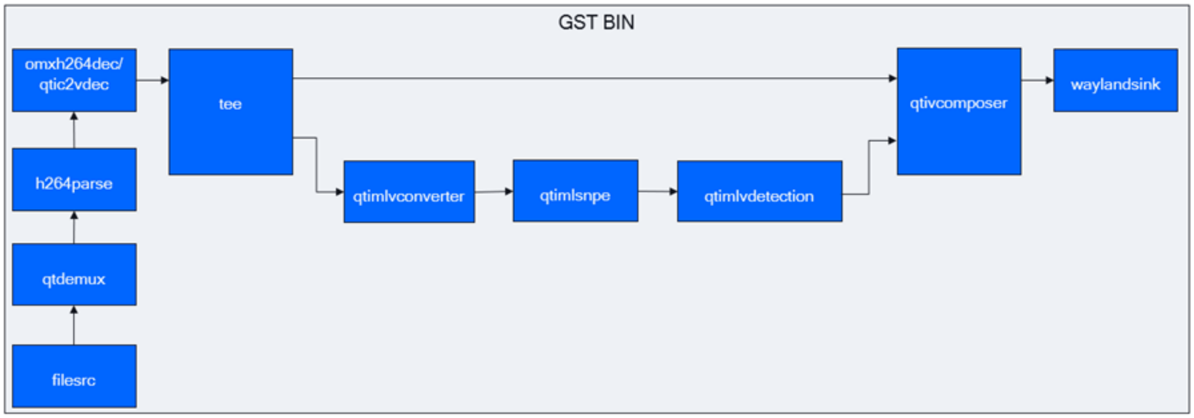


Figure 3-54 Single H264 file stream with object detection using video overlay and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue !
qtioverlay ! queue ! waylandsink sync=false fullscreen=true \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse !  qtic2vdec ! queue ! tee name=split ! queue ! metamux. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection threshold=75.0
results=10 module=yolov5m labels=/data/yolov5m.labels ! text/x-raw ! queue !
metamux
```

3.22 qtimlvclassification

The qtimlvclassification element processes output tensors of an image classification model from ML inference plug-in (such as qtimlTFLite, qtimlsnpe and qtimlaic) into result of predictions.

The negotiated GstCaps determines the processed output on the plug-in output. It can be either of the following:

- An image mask (GstCaps: video/x-raw), which can be applied over the original image using qtivcomposer.
- A GStreamer formatted text (GstCaps: text/x-raw) containing the prediction results.

qtimlvclassification leverages the CPU-based Cairo 2D graphics library for image overlay mask. This enables it to draw the prediction results in ION/DMA buffers, which are allocated by the GstImageBufferPool custom buffer pool class through IOCTL commands to the kernel.

In the versatile text format, the prediction results are parsed into GStreamer-formatted string inside the buffers allocated using the regular system memory.

The module and labels properties of the plug-in determine the method used for postprocessing operations.

- The module property specifies the post-process module, which is populated and runs dynamically at run-time with the available libraries at `in /usr/lib/gstreamer-1.0/ml/modules/` containing the prefix `ml-vclassification-`
- The labels property is a customized text file different for each machine learning detection model that you need to provide for the prediction labels.

Optional properties are available for adjusting the prediction results. Use results to control the number of results displayed and use threshold to set a confidence threshold for prediction, results with confidence below the threshold are not displayed.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstMLVideoClassification

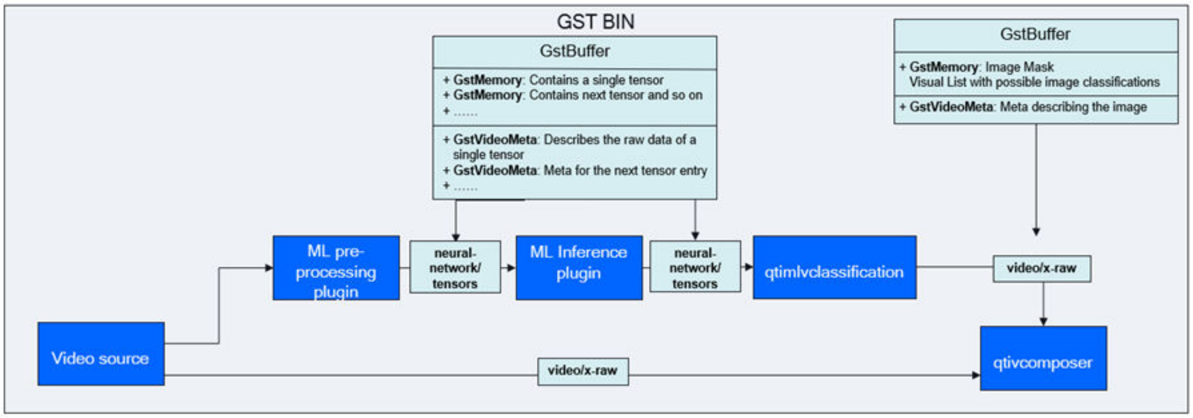

GST BIN
GstBuffer
+ GstMemory: Contains a single tensor
+ GstMemory: Contains next tensor and so on
+ ......
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ......
GstBuffer
+ GstMemory: Image Mask Visual List with possible image classifications
+ GstVideoMeta: Meta describing the image
ML pre-processing plugin
neural-network/tensors
ML Inference plugin
neural-network/tensors
qtimlvclassification
video/x-raw
Video source
video/x-raw
qtivcomposer

Figure 3-55 Postprocessing for object detection architecture

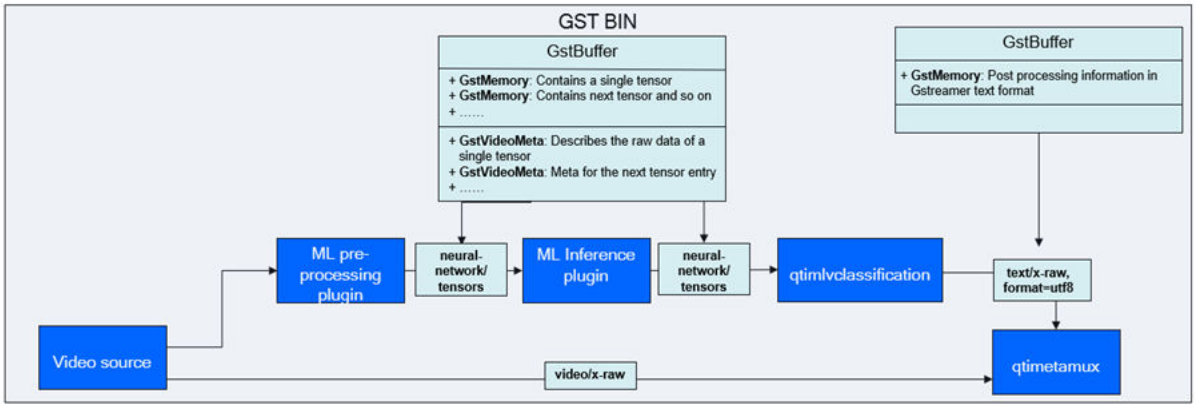

GST BIN
GstBuffer
+ GstMemory: Contains a single tensor
+ GstMemory: Contains next tensor and so on
+ ......
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ......
GstBuffer
+ GstMemory: Post processing information in Gstreamer text format
ML pre-processing plugin
neural-network/tensors
ML Inference plugin
neural-network/tensors
qtimlvclassification
text/x-raw, format=utf8
Video source
video/x-raw
qtimetamux

Figure 3-56 qtimlvclassification in pipeline

qtimlvclassification pad configuration

Table 3-47 Pad templates for qtimlvclassification

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* On request *Direction:* sink	neural-network/tensors	–	–
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-raw	format:	{ (string)BGRA, (string)BGRx, (string)BGR16 }
	video/x-raw(memory:GBM)	format:	{ (string)BGRA, (string)BGRx, (string)BGR16 }
	text/x-raw	format:	{ (string)utf8 }

qtimlvclassification element configuration

Table 3-48 Element properties of qtimlvclassification

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "mlvideoclassification0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
qos	■ Handle Quality-of-Service events ■ flags: readable, writable ■ Boolean. Default: false
module	Module name that is going to be used for processing the tensors flags: readable, writable Enum "GstMLVideoClassificationModules" Default: 0, "none" ■ (0): none - No module, default invalid mode ■ (1): MobileNet - ml-vclassification-MobileNet
labels	■ Labels filename ■ flags: readable, writable ■ String. Default: null
results	■ Number of results to display ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 10 Default: 5
threshold	■ Confidence threshold in % ■ flags: readable, writable ■ Double. Range: 10.0 - 100.0 Default: 10.0

Usage

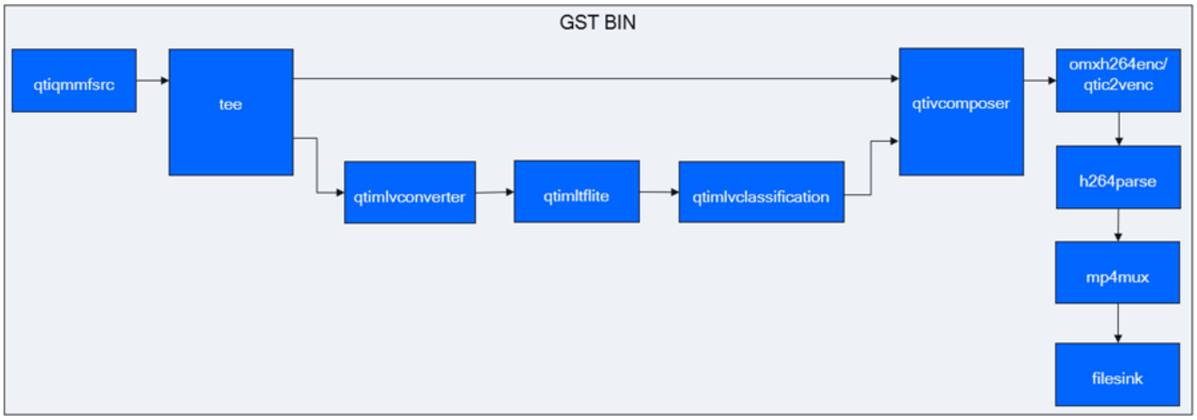


Figure 3-57 Single camera stream with image classification using video composer and saved to file

Command:

```
st-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer sink_1::position="<50,
50>" sink_1::dimensions="<368, 64>" ! queue ! qtic2venc ! h264parse ! queue !
mp4mux ! queue ! filesink location=/data/video.mp4 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/mobilenet_v2_1.0_224_quant.tflite ! queue ! qtimlvclassification
threshold=60.0 results=3 module=mobilenet labels=/data/mobilenet.labels !
video/x-raw,format=BGRA,width=368,height=64 ! queue ! mixer
```

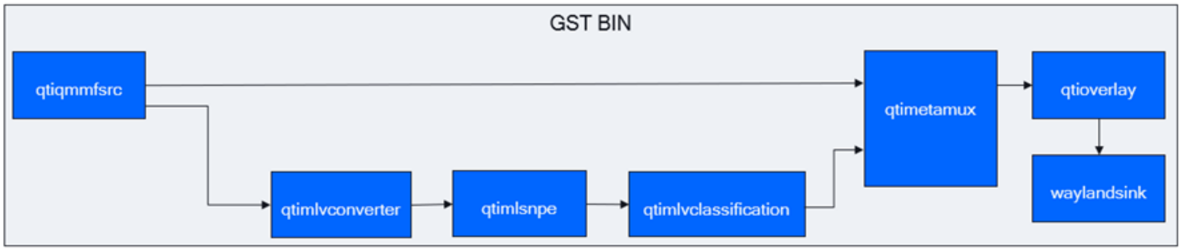


Figure 3-58 High and low-resolution camera streams with image classification using video overlay and displayed on screen

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue ! qtioverlay
text-font-size=24 ! queue ! waylandsink sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! metamux. \
camrsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=640,height=360,framerate=30/1 ! queue !
qtimlvconverter ! queue ! qtimlsnpe delegate=dsp model=/data/
mobilenet_v1_quantaware_quantized.dlc ! queue ! \
qtimlvclassification threshold=51.0 results=1 module=mobilenet labels=/data/
mobilenet.labels ! text/x-raw ! queue ! metamux
```

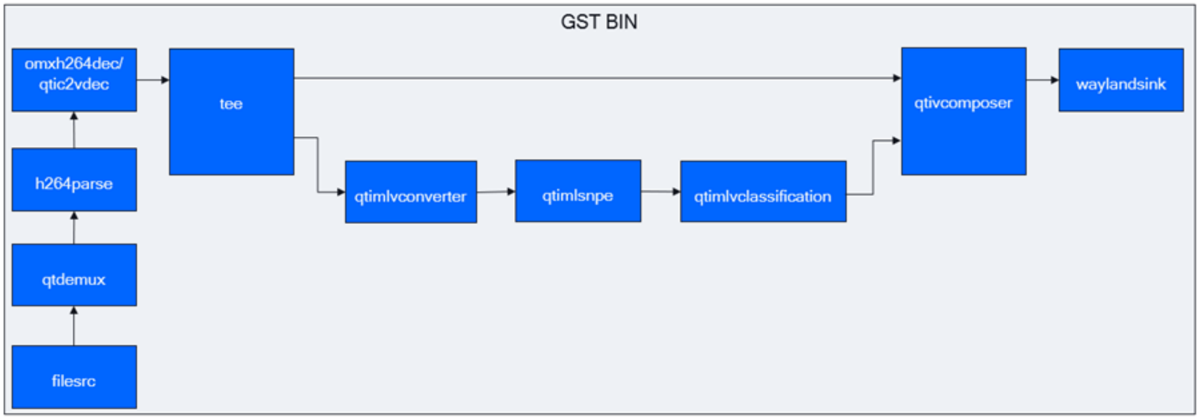


Figure 3-59 Single H264 file stream with image classification using video composer and displayed on screen

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::position="<50, 50>" sink_1::dimensions="<368, 32>" ! queue !
waylandsink sync=true fullscreen=true \
filesrc location=/data/Animals_000_720p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse !  qtic2vdec ! queue ! tee name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=dsp model=/data/
resnet50_enhanced_quantized.dlc ! queue ! qtimlvclassification threshold=51.0
results=1 module=mobilenet labels=/data/resnet50.labels ! video/x-
raw,format=BGRA,width=368,height=32 ! queue ! mixer
```

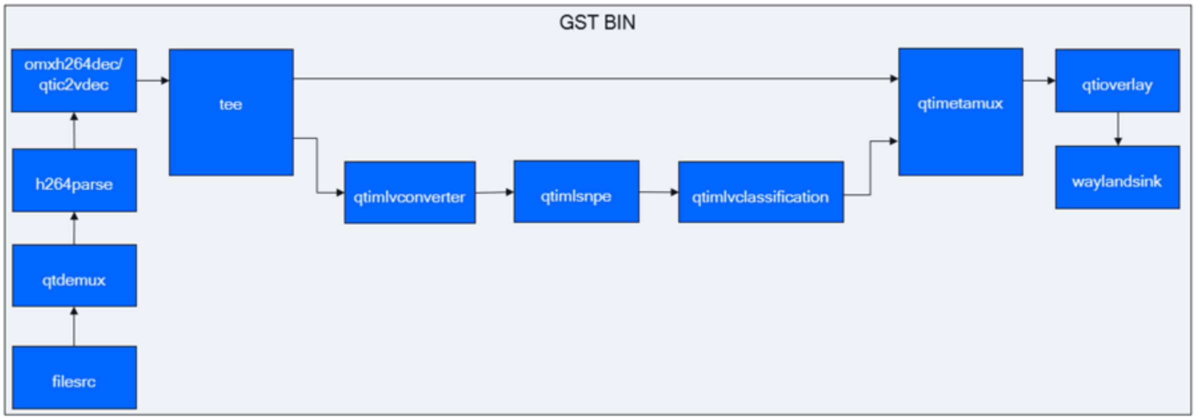


Figure 3-60 Single H264 file stream with image classification using video overlay and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue ! qtioverlay
text-font-size=24 ! queue ! waylandsink sync=true fullscreen=true \
filesrc location=/data/Animals_003_720p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse !   qtic2vdec ! queue ! tee name=split ! queue ! metamux. \
split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=dsp model=/data/
resnet50_enhanced_quantized.dlc ! queue ! qtimlvclassification threshold=51.0
results=1 module=mobilenet labels=/data/resnet50.labels ! text/x-raw !
queue ! metamux
```

3.23 qtimlvsegmentation

The qtimlvsegmentation element processes output tensors of an image segmentation/depth estimation model from ML inference plug-in (such as qtimltflite, qtimlsnpe and qtimlaic) into result of predictions.

The processed output is an image mask (GstCaps: video/x-raw) with dimensions and format determined by the negotiated plug-in output GstCaps and which can be applied over the original image using qtivcomposer.

For this mask, the element will leverage the CPU based Cairo 2D graphics library to draw the prediction results in ION/DMA buffers allocated by the custom buffer pool class GstImageBufferPool through IOCTL commands to the kernel.

The method used for this postprocessing operations is determined by the module and labels properties of the plug-in. The module property specifies which post-process module to run and is populated dynamically at run time with the libraries available in `/usr/lib/gstreamer-1.0/ml/modules/` containing the prefix `ml-vsegmentation-` and the labels property is a customized text file different for each machine learning detection model that you need to provide for the prediction labels.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstMLVideoSegmentation

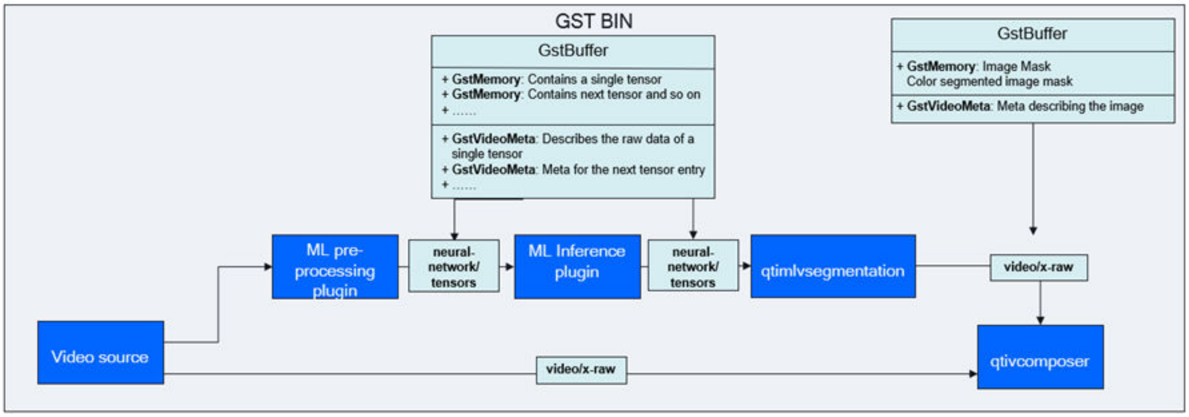

GST BIN
GstBuffer
+ GstMemory: Contains a single tensor
+ GstMemory: Contains next tensor and so on
+ ......
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ......
GstBuffer
+ GstMemory: Image Mask Color segmented image mask
+ GstVideoMeta: Meta describing the image
ML pre-processing plugin
neural-network/tensors
ML Inference plugin
neural-network/tensors
qtimlvsegmentation
video/x-raw
Video source
video/x-raw
qtivcomposer

Figure 3-61 qtimlvsegmentation in GStreamer pipeline

qtimlvsegmentation pad configuration

Table 3-49 Pad templates for qtimlvsegmentation

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* On request *Direction:* sink	neural-network/tensors		
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-raw	format:	{ (string)BGRA, (string)BGRx, (string)BGR16 }
	text/x-raw	format:	{ (string)utf8 }

qtimlvsegmentation element configuration

Table 3-50 Element properties for qtimlvsegmentation

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "mlvideosegmentation0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"

Table 3-50 Element properties for qtimlvsegmentation (cont.)

Property	Description
qos	▪ Handle Quality-of-Service events ▪ flags: readable, writable ▪ Boolean. Default: false
module	▪ Module name that is going to be used for processing the tensors ▪ flags: readable, writable Enum "GstMLVideoSegmentationModules" Default: 0, "none" ▪ (0): none - No module, default invalid mode ▪ (1): deeplab-argmax - ml-vsegmentation-deeplab-argmax ▪ (2): midas-v2 - ml-vsegmentation-midas-v2
labels	▪ Labels filename ▪ flags: readable, writable ▪ String. Default: null

Usage

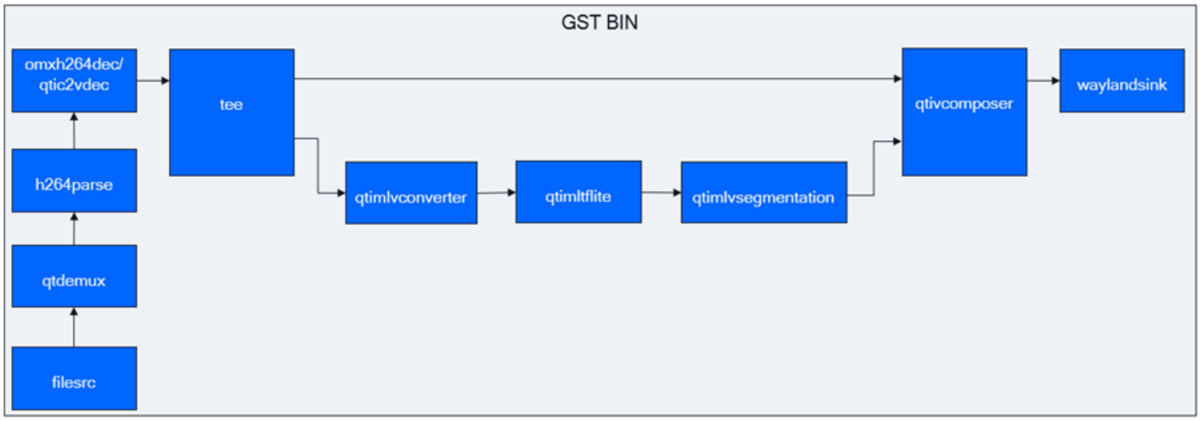


Figure 3-62 Single H264 file stream with image segmentation using video composer and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::dimensions="<1920,1080>" sink_1::alpha=0.5 ! queue ! waylandsink
sync=false fullscreen=true \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse !  qtic2vdec ! queue ! tee name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/dv3_argmax_int32.tflite ! queue ! qtimlvsegmentation module=deeplab-
argmax labels=/data/dv3-argmax.labels ! video/x-raw,width=256,height=144 !
queue ! mixer
```

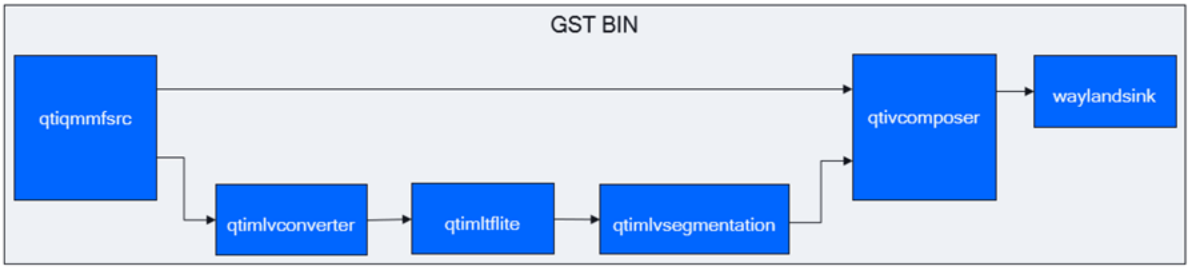


Figure 3-63 High and low-resolution camera streams with image segmentation using video composer and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::dimensions="<1920,1080>" sink_1::alpha=0.5 ! queue ! waylandsink
sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! metamux. \
camrsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=640,height=360,framerate=30/1 ! queue !
qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/data/
dv3_argmax_int32.tflite ! queue ! \
qtimlvsegmentation module=deeplab-argmax labels=/data/dv3-argmax.labels !
video/x-raw,width=256,height=144 ! queue ! mixer
```

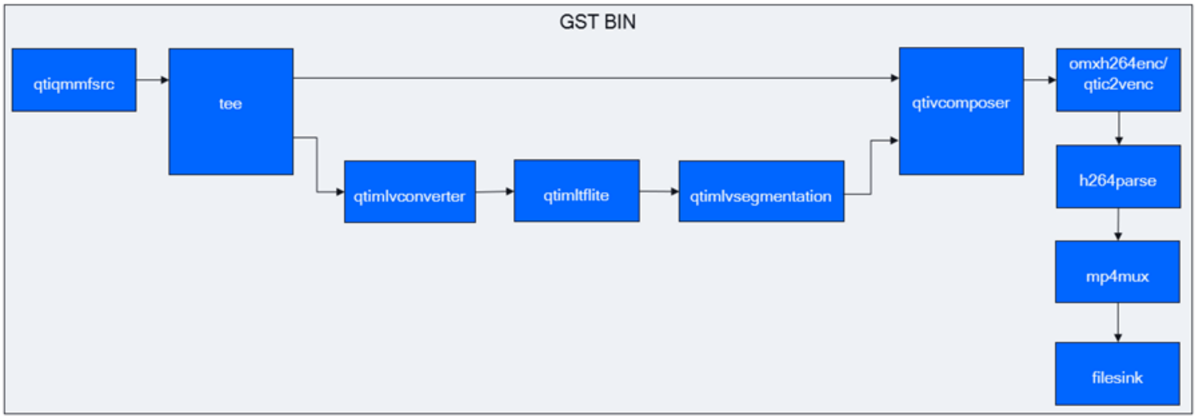


Figure 3-64 Single camera stream with image segmentation using video composer and saved to file

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::dimensions="<1920,1080>" sink_1::alpha=0.5 ! queue ! qtic2venc !
h264parse ! queue ! mp4mux ! queue ! filesink location=/data/video.mp4 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/dv3_argmax_int32.tflite ! queue ! qtimlvsegmentation module=deeplab-
```

```
argmax labels=/data/dv3-argmax.labels ! video/x-raw,width=256,height=144 !
queue ! mixer
```

3.24 qtimlvpose

The qtimlvpose element processes output tensors of an Pose Estimation model from ML inference plug-in (such as qtimltflite, qtimlsnpe and qtimlaic) into result of predictions.

The processed output is determined by the negotiated GstCaps on the plug-in output. It can be either an image mask (GstCaps: video/x-raw) which for an example can be applied over the original image using qtivcomposer or GStreamer formatted text (GstCaps: text/x-raw) containing the prediction results

For image overlay mask the element will leverage the CPU based Cairo 2D graphics library to draw the prediction results in ION/DMA buffers allocated by the custom buffer pool class GstImageBufferPool through IOCTL commands to the kernel. While in the versatile text format the prediction results will be parsed into GStreamer formatted string inside buffers allocated using regular system memory

The method used for this postprocessing operations is determined by the module and labels properties of the plug-in. The module property specifies which post-process module to run and is populated dynamically at run time with the libraries available in `/usr/lib/gstreamer-1.0/ml/modules/` containing the prefix "`ml-vpose`-" and the labels property is a customized text file different for each machine learning detection model that user needs to provide for the prediction labels.

Optional properties are available for adjusting the prediction results. Use results to control the number of results displayed and use threshold to set a confidence threshold for prediction, results with confidence below that won't be displayed.

Inheritance chain: GObject → GstObject → GstElement → GstBaseTransform → GstMLVideoPose

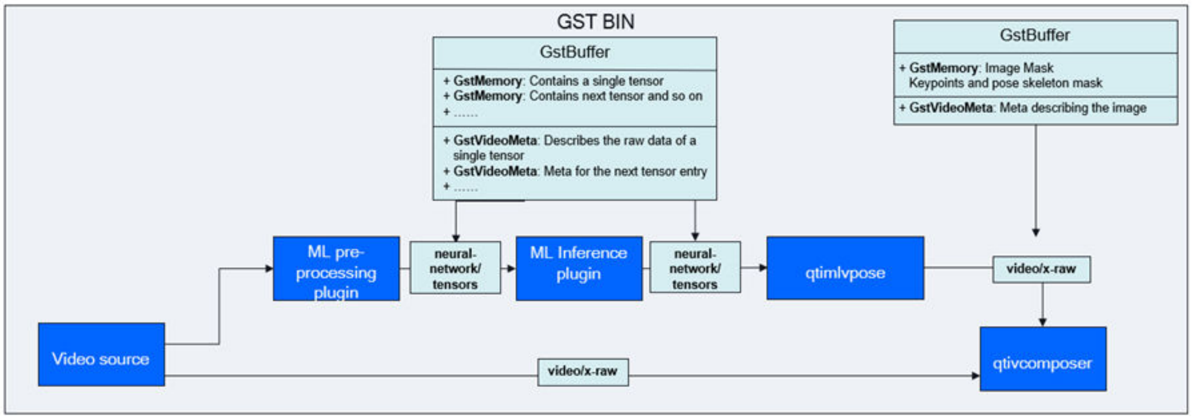

GST BIN
GstBuffer
+ GstMemory: Contains a single tensor
+ GstMemory: Contains next tensor and so on
+ ......
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ......
GstBuffer
+ GstMemory: Image Mask Keypoints and pose skeleton mask
+ GstVideoMeta: Meta describing the image
ML pre-processing plugin
neural-network/tensors
ML Inference plugin
neural-network/tensors
qtimlvpose
video/x-raw
Video source
video/x-raw
qtivcomposer

Figure 3-65 Postprocessing for pose estimation architecture

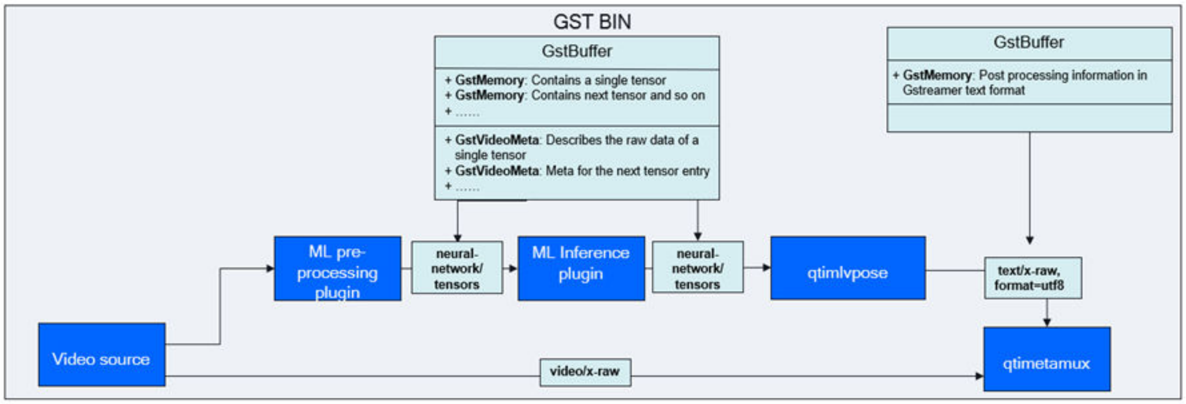

GST BIN
GstBuffer
+ GstMemory: Contains a single tensor
+ GstMemory: Contains next tensor and so on
+ ......
+ GstVideoMeta: Describes the raw data of a single tensor
+ GstVideoMeta: Meta for the next tensor entry
+ ......
GstBuffer
+ GstMemory: Post processing information in Gstreamer text format
ML pre-processing plugin
neural-network/tensors
ML Inference plugin
neural-network/tensors
qtimlvpose
text/x-raw, format=utf8
Video source
video/x-raw
qtimetamux

Figure 3-66 qtimlvpose in ML pipeline

qtimlvpose pad configuration

Table 3-51 Pad templates for qtimlvpose

Pad Name	Capabilities		
SINK template: 'sink' *Availability:* On request *Direction:* sink	neural-network/tensors	–	–
SRC template: 'src' *Availability:* Always *Direction:* source	video/x-raw	format:	{ (string)BGRA, (string)BGRx, (string)BGR16 }
	text/x-raw	format:	{ (string)utf8 }

qtimlvpose element configuration

Table 3-52 Element properties of qtimlvpose

Property	Description
name	■ The name of the object ■ flags: readable, writable ■ String. Default: "mlvideopose0"
parent	■ The parent of the object ■ flags: readable, writable ■ Object of type "GstObject"
qos	■ Handle Quality-of-Service events ■ flags: readable, writable ■ Boolean. Default: false
module	■ Module name that is going to be used for processing the tensors ■ flags: readable, writable Enum "GstMLVideoPoseModules" Default: 0, "none" ■ (0): none - No module, default invalid mode ■ (1): posenet - ml-vpose-posenet

Table 3-52 Element properties of qtimlvpose (cont.)

Property	Description
labels	■ Labels filename ■ flags: readable, writable ■ String. Default: null
results	■ Number of results to display ■ flags: readable, writable ■ Unsigned Integer. Range: 0 - 10 Default: 5
threshold	■ Confidence threshold in % ■ flags: readable, writable ■ Double. Range: 10.0 - 100.0 Default: 50.0

Usage

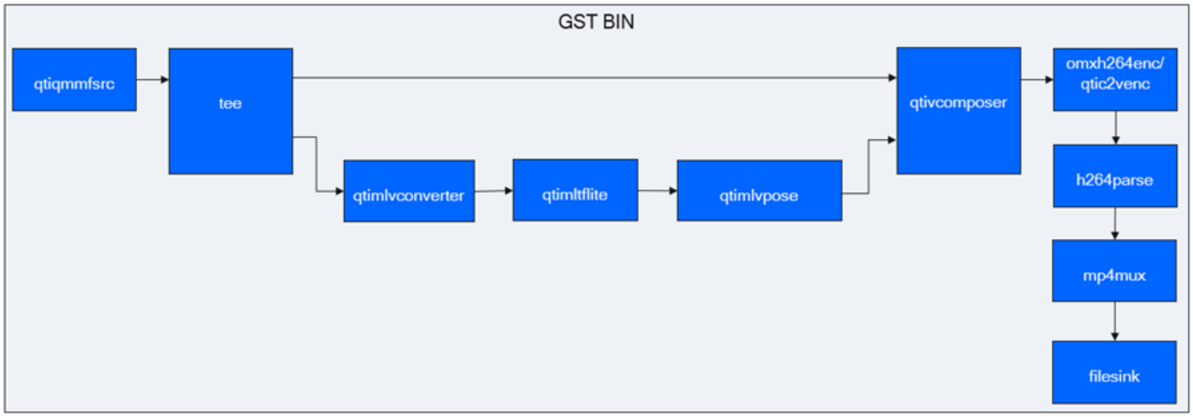


Figure 3-67 Single camera stream with pose estimation using video composer and saved to file

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtivcomposer name=mixer
sink_1::dimensions="<1920,1080>" ! queue ! qtic2venc ! h264parse ! queue !
mp4mux ! queue ! filesink location=/data/video.mp4 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! mixer. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/posenet_MobileNet_v1_075_481_641_quant.tflite ! queue ! \
qtimlvpose threshold=51.0 results=2 module=posenet labels=/data/
posenet.labels ! video/x-raw,format=BGRA,width=640,height=360 ! queue ! mixer
```

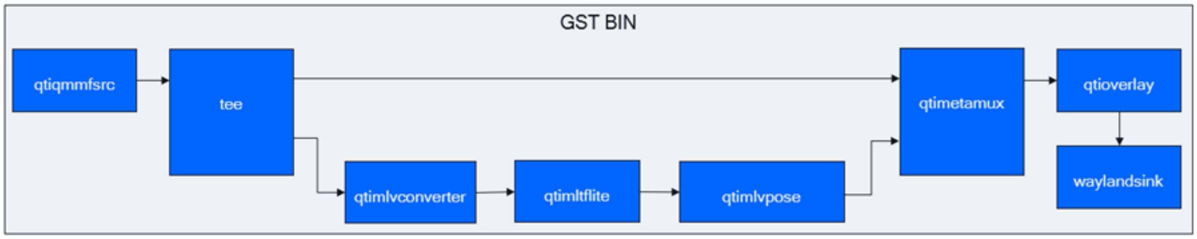


Figure 3-68 Single camera stream with pose estimation using video overlay and displayed on screen

Command to connect to the display:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue !
qtioverlay ! queue ! waylandsink sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! metamux. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/posenet_MobileNet_v1_075_481_641_quant.tflite ! queue ! \
qtimlvpose threshold=51.0 results=2 module=posenet labels=/data/
posenet.labels ! text/x-raw ! queue ! metamux
```

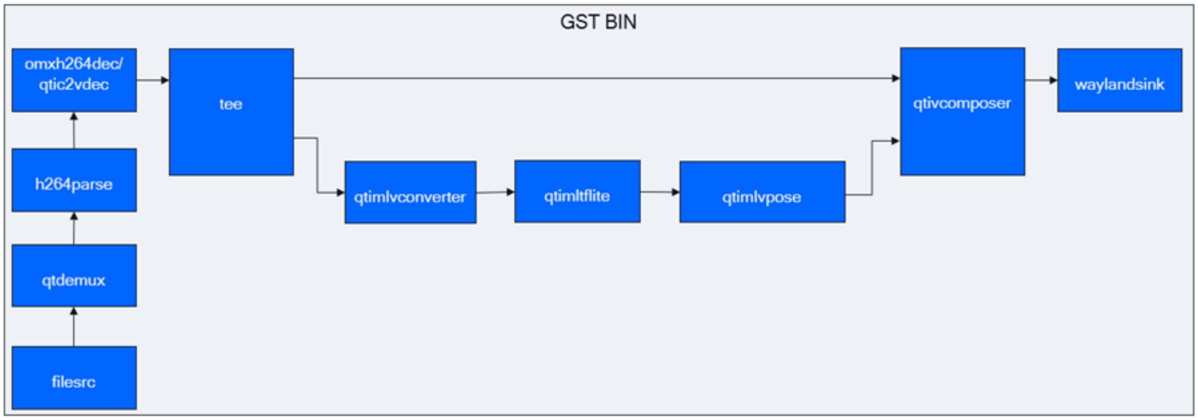


Figure 3-69 Single H264 file stream with pose estimation using video overlay and displayed on screen

Command to connect to display:

```
gst-launch-1.0 -e --gst-debug=2 qtimetamux name=metamux ! queue !
qtioverlay ! queue ! waylandsink sync=false fullscreen=true \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! metamux. \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/posenet_MobileNet_v1_075_481_641_quant.tflite ! queue ! \
qtimlvpose threshold=51.0 results=2 module=posenet labels=/data/
posenet.labels ! text/x-raw ! queue ! metamux
```

4 QIM SDK base utilities

The GStreamer framework base provides most of the backbone classes needed for implementation of custom plug-in solutions with most of them having CPU based implementations and do not understand the Qualcomm-specific hardware. To leverage hardware acceleration in IM SDK and enable the tensor-based approach for ML, custom classes and layers are required.

This section describes the various custom implementations that are part of the IM SDK and can be used as building blocks for any plug-in and GStreamer applications.

4.1 Video APIs

The video APIs enable you to configure the video buffers and video converter.

GstImageBufferPool APIs

GstImageBufferPool is a special GstBufferPool subclass for raw video buffers. It allows configuration of video-specific requirements such as stride alignments or pixel padding, type of memory to use (GBM / ION), and can also be configured to automatically add GstVideoMeta to the buffers.

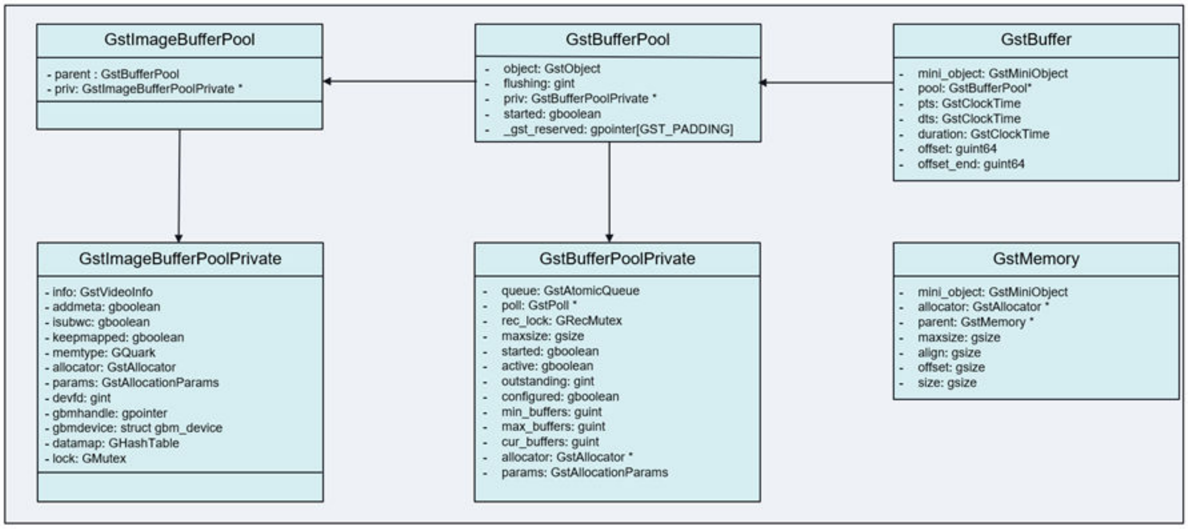


Figure 4-1 GstImageBufferPool APIs

- `gst_image_buffer_pool_new()`
  - **Description**

    Create a new bufferpool that can allocate video frames.

    ```
    GstBufferPool *
    gst_image_buffer_pool_new (const gchar * type)
    ```
  - **Parameters**

    `type` – the memory that the pool will use for allocating buffers
  - **Returns**
    - New GstBufferPool to allocate ML frames
    - Free with gst_object_unref

The following are buffer memories for `gst_image_buffer_pool_new()`:

- GST_IMAGE_BUFFER_POOL_TYPE_ION

  A possible memory type that the pool uses for allocating buffers by passing it as an argument to gst_ml_buffer_pool_new.

  ```
  #define GST_IMAGE_BUFFER_POOL_TYPE_ION "GstBufferPoolTypeIonMemory"
  ```
- GST_IMAGE_BUFFER_POOL_TYPE_GBM

  A possible memory type that the pool will use for allocating buffers by passing it as an argument to gst_ml_buffer_pool_new.

  ```
  #define GST_IMAGE_BUFFER_POOL_TYPE_GBM "GstBufferPoolTypeGbmMemory"
  ```
- GST_IMAGE_BUFFER_POOL_OPTION_UBWC_MODE

  An option indicating that the allocated buffer must be UBWC.

  ```
  #define GST_IMAGE_BUFFER_POOL_OPTION_UBWC_MODE
  "GstBufferPoolOptionUBWCMode"
  ```
- GST_IMAGE_BUFFER_POOL_OPTION_KEEP_MAPPED

  An option indicating that once the buffer memory is mapped it will be kept mapped until the memory is destroyed.

  ```
  #define GST_IMAGE_BUFFER_POOL_OPTION_KEEP_MAPPED
  "GstBufferPoolOptionKeepMapped"
  ```

GstGlesVideoConverter

This object leverages the GPU through the OpenGLES based IB2C library to provide hardware-accelerated image manipulation, transformation, and color conversion operations.

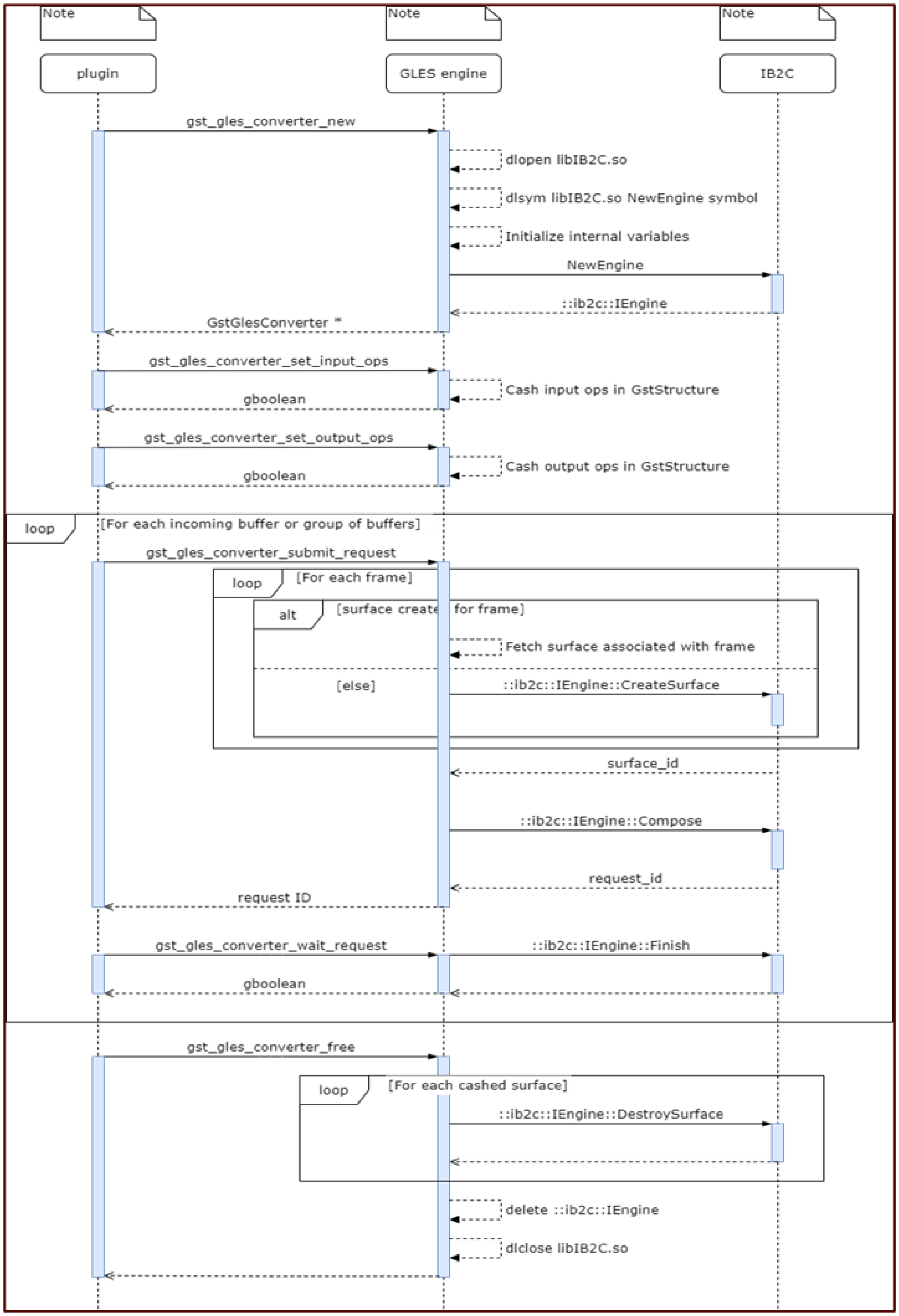

Note
Note
Note
plugin
GLES engine
IB2C
gst_gles_converter_new
dlopen libIB2C.so
dlsym libIB2C.so NewEngine symbol
Initialize internal variables
NewEngine
::ib2c::IEngine
GstGlesConverter *
gst_gles_converter_set_input_ops
Cash input ops in GstStructure
gboolean
gst_gles_converter_set_output_ops
Cash output ops in GstStructure
gboolean
loop
[For each incoming buffer or group of buffers]
gst_gles_converter_submit_request
loop
[For each frame]
alt
[surface create for frame]
Fetch surface associated with frame
[else]
::ib2c::IEngine::CreateSurface
surface_id
::ib2c::IEngine::Compose
request_id
request ID
gst_gles_converter_wait_request
::ib2c::IEngine::Finish
gboolean
gst_gles_converter_free
loop
[For each cashed surface]
::ib2c::IEngine::DestroySurface
delete ::ib2c::IEngine
dlclose libIB2C.so

Figure 4-2 Hardware-accelerated image manipulation, transformation, and color conversion

The following are the GstGlesVideoConverter APIs:

- `gst_gles_video_converter_new()`
  - **Description**

    Create a new GstGlesVideoConverter.

    ```
    GstGlesVideoConverter *
    gst_gles_video_converter_new (void)
    ```

  - **Returns**

    A new GstGlesVideoConverter, free with `gst_gles_video_converter_free()`.

- `gst_gles_video_converter_free()`
  - **Description**

    Release all resources held by the converter and free the memory associated with it.

    ```
    void
    gst_gles_video_converter_new (GstGlesVideoConverter * convert)
    ```

  - **Parameters**

    `convert`–A GstGlesVideoConverter

  - **Returns**–None

- `gst_gles_video_converter_submit_request()`
  - **Description**

    Submit a number of video composition which will be executed together.

    ```
    gpointer
    gst_gles_video_converter_submit_request (GstGlesVideoConverter
    *convert, GstGlesComposition * compositions, guint n_compositions)
    ```

  - **Parameters**
    - `convert`–A GstGlesVideoConverter
    - `compositions` –An array of composition frames
    - `n_compositions`–The number of compositions

  - **Returns**

    Unique pointer request ID if the operation was successful.

- `gst_gles_video_converter_wait_request()`
  - **Description**

    Wait for the submitted to the GPU compositions to finish.

    ```
    gboolean
    gst_gles_video_converter_wait_request (GstGlesVideoConverter *convert,
    gpointer request_id)
    ```

- **Parameters**
  - `convert`–A GstGlesVideoConverter
  - `request_id`–The request ID
- **Returns**

  TRUE if the operation was successful.

- `gst_gles_video_converter_flush()`
  - **Description**

    Wait for compositions submitted to the GPU to finish.

    ```
    void
    gst_gles_video_converter_flush (GstGlesVideoConverter *convert)
    ```
  - **Parameters**

    `convert`–A GstGlesVideoConverter
  - **Returns**

    None

4.2 Machine learning APIs

4.2.1 GstMLType

The following are the possible values describing the tensor format:

Enumeration	Description
GST_ML_TYPE_UNKNOWN	Invalid data
GST_ML_TYPE_INT8	Data is represented as 1 byte of signed integer value
GST_ML_TYPE_UINT8	Data is represented as 1 byte of unsigned integer value
GST_ML_TYPE_INT32	Data is represented as 4 byte of signed integer value
GST_ML_TYPE_UINT32	Data is represented as 4 byte of unsigned integer value
GST_ML_TYPE_FLOAT16	Data is represented as 2 bytes of floating-point value
GST_ML_TYPE_FLOAT32	Data is represented as 4 bytes of floating-point value

- `gst_ml_type_get_size()`
  - **Description**

    Returns the size of the GstMLType in bytes

    ```
    guint
    gst_ml_type_get_size (GstMLType type)
    ```
  - **Parameters**

    `type`–A GstMLType
  - **Returns**

    The size in bytes.

- `gst_ml_type_from_string()`
  - **Description**

    Returns GstMLType based on its string version.

    ```
    GstMLType
    gst_ml_type_from_string (const gchar * type)
    ```
  - **Parameters**

    `type`–A char string version of a GstMLType
  - **Returns**

    A GstMLType
- `gst_ml_type_to_string()`
  - **Description**

    Returns a char string version of a GstMLType

    ```
    const gchar *
    gst_ml_type_to_string (GstMLType type)
    ```
  - **Parameters**

    `type`–A string version of a GstMLType
  - **Returns**

    A new char string version of a GstMLType

4.2.2 GstMLInfo

GstMLInfo represents the information structure describing the machine learning properties. This information can be filled in from GstCaps with `gst_ml_info_from_caps()`.

Table 4-1 Information structure describing ML properties

Field	Description
`type (GstMLType)`	Type of the tensors
`n_tensors (guint)`	Number of tensors
`n_dimensions (guint)`	Number of dimensions for each tensor
`tensors (guint)`	Array with tensor dimensions

- `gst_ml_info_init()`
  - **Description**

    Initializes the GstMLInfo fields

    ```
    void
    gst_ml_info_init (GstMLInfo * info)
    ```

- **Parameters**

  `info`–A GstMLInfo

- **Returns**

  None

- `gst_ml_info_new()`

  - **Description**

    Copy a GstMLInfo structure

    ```
    GstMLInfo *
    gst_ml_info_new (void)
    ```

  - **Returns**

    A new GstMLInfo. Free with gst_ml_info_free.

- `gst_ml_info_copy()`

  - **Description**

    Copy a GstMLInfo structure

    ```
    GstMLInfo *
    gst_ml_info_copy (const GstMLInfo * info)
    ```

  - **Parameters**

    `info`–A GstMLInfo

  - **Returns**

    A new GstMLInfo. Free with gst_ml_info_free.

- `gst_ml_info_free()`

  - **Description**

    Free a GstMLInfo structure previously allocated with `gst_ml_info_new()` or `gst_ml_info_copy()`.

    ```
    void
    gst_ml_info_free (const GstMLInfo * info)
    ```

  - **Parameters**

    `info` –A GstMLInfo

  - **Returns**

    None

- `gst_ml_info_from_caps()`

  - **Description**

    Parse caps and update info

    ```
    gboolean
    gst_ml_info_from_caps (GstMLInfo * info, const GstCaps  * caps)
    ```

- **Parameters**
  - `info`–A GstMLInfo
  - `caps`–A GstCaps
- **Returns**

  TRUE if the operation was successful

- `gst_ml_info_to_caps()`
  - **Description**

    Convert the values of info into a GstCaps

    ```
    GstCaps *
    gst_ml_info_to_caps (const GstMLInfo * info)
    ```

  - **Parameters**

    `info`–A GstMLInfo

  - **Returns**

    A new GstCaps containing the information

- `gst_ml_info_is_equal()`
  - **Description**

    Compares two GstMLInfo and returns whether they are equal or not.

    ```
    gboolean
    gst_ml_info_is_equal (const GstMLInfo * l_info, const GstMLInfo *
    r_info)
    ```

  - **Parameters**
    - l_info–A GstMLInfo
    - r_info–A GstMLInfo
  - **Returns**

    TRUE if the operation was successful

- `gst_ml_info_tensor_size()`
  - **Description**

    Calculates the size of the tensor specified by its index from GstMLInfo

    ```
    gsize
    gst_ml_info_tensor_size (const GstMLInfo * info, guint index)
    ```

  - **Parameters**
    - `info`–A GstMLInfo
    - `index`–The tensor index
  - **Returns**

    The size in bytes.

- `gst_ml_info_size()`
  - **Description**

    Calculates the total size of all tensors inside GstMLInfo

    ```
    gsize
    gst_ml_info_size (const GstMLInfo * info)
    ```
  - **Parameters**

    `info`– A GstMLInfo
  - **Returns**

    The size in bytes.

4.2.3 GstMLTensorMeta

Field	Description
meta (GstMeta)	Parent GstMeta
id (guint)	ID corresponding to the memory index inside GstBuffer
type (GstMLType)	Tensor type
n_dimensions (guint)	Number of tensor dimensions
dimensions (guint)	Array of tensor dimensions

- `gst_buffer_add_ml_tensor_meta()`
  - **Description**

    Attaches GstMLTensorMeta metadata to buffer with the given parameters.

    ```
    GstMLTensorMeta *
    gst_buffer_add_ml_tensor_meta (GstBuffer * buffer, const GstMLType
    type, const guint n_dimensions, const guint
    dimensions[GST_ML_TENSOR_MAX_DIMS])
    ```
  - **Parameters**
    - `buffer`–A GstBuffer
    - `type`–The tensor type
    - `n_dimensions`–The number of tensor dimensions
    - `dimensions`–The array containing the tensor dimensions
  - **Returns**

    The GstMLTensorMeta on buffer. Do not free after the code is run.
- - `gst_buffer_get_ml_tensor_meta()`
    - **Description**

      Get the first GstMLTensorMeta on buffer

      ```
      GstMLTensorMeta *
      gst_buffer_get_ml_tensor_meta (GstBuffer * buffer)
      ```

- **Parameters**

  `buffer`–A GstBuffer

- **Returns**

  The GstMLTensorMeta on buffer. Do not free after the code is run.

- ▫ `gst_buffer_get_ml_tensor_meta_id()`
  - **Description**

    Find the GstMLTensorMeta on buffer with the given ID

    ```
    GstMLTensorMeta *
    gst_buffer_get_ml_tensor_meta_id (GstBuffer * buffer, guint id)
    ```

  - **Parameters**

    `buffer`–A GstBuffer

    `id`–A metadata ID

  - **Returns**

    The GstMLTensorMeta on buffer. Do not free after the code is run.

- ▫ `gst_ml_meta_tensor_size()`
  - **Description**

    Get the total size of the tensor in bytes, calculated based of the dimensions and tensor type.

    ```
    gsize
    gst_ml_meta_tensor_size (const GstMLTensorMeta * meta)
    ```

  - **Parameters**

    meta–A GstMLTensorMeta

  - **Returns**

    The size in bytes.

4.2.4 GstMLFrame

GstMLFrame is a convenient structure obtained from `gst_ml_frame_map()`, containing mapped pointers to the tensors and all the necessary information about them.

Table 4-2 Structure of GstMLFrame

Field	Description
info (GstMLInfo)	The GstMLInfo
buffer (GstBuffer)	Mapped buffer containing the tensor memory blocks
map (GstMapInfo)	Mappings of the tensor memory blocks

- `gst_ml_frame_map()`
  - **Description**

    Use information and buffer to fill in the values of frame. frame is usually allocated on the stack, and you will pass the address to the GstMLFrame structure allocated on the stack. The function will then fill in the structures with the various ML tensor-specific information you need to access the data of the ML buffer. All video tensors of buffer will be mapped, and the pointers will be set in frame->data.

```
gboolean
gst_ml_frame_map (GstMLFrame * frame, const GstMLInfo * info, GstBuffer
* buffer, GstMapFlags flags)
```

- **Parameters**
  - `frame`–pointer to GstMLFrame
  - `info`–AGstMLInfo
  - `buffer`–A GstBuffer
  - `flags`–A GstMapFlags
- **Returns**

  TRUE if the map operation was successful

- `gst_ml_frame_unmap()`
  - **Description**

    Unmap the memory that was previously mapped with gst_ml_frame_map

    ```
    void
    gst_ml_frame_unmap (GstMLFrame * frame)
    ```

  - **Parameters**

    `frame`–pointer to GstMLFrame

  - **Returns**

    None

4.2.5 GstMLBufferPool

GstMLBufferPool is a special GstBufferPool subclass for ML tensor buffers. It allows the configuration of tensor-specific requirements such as multiple GstMemory blocks in single GstBuffer

The type of memory used for allocation and can also be configured to automatically add GstMLTensorMeta to the buffers.

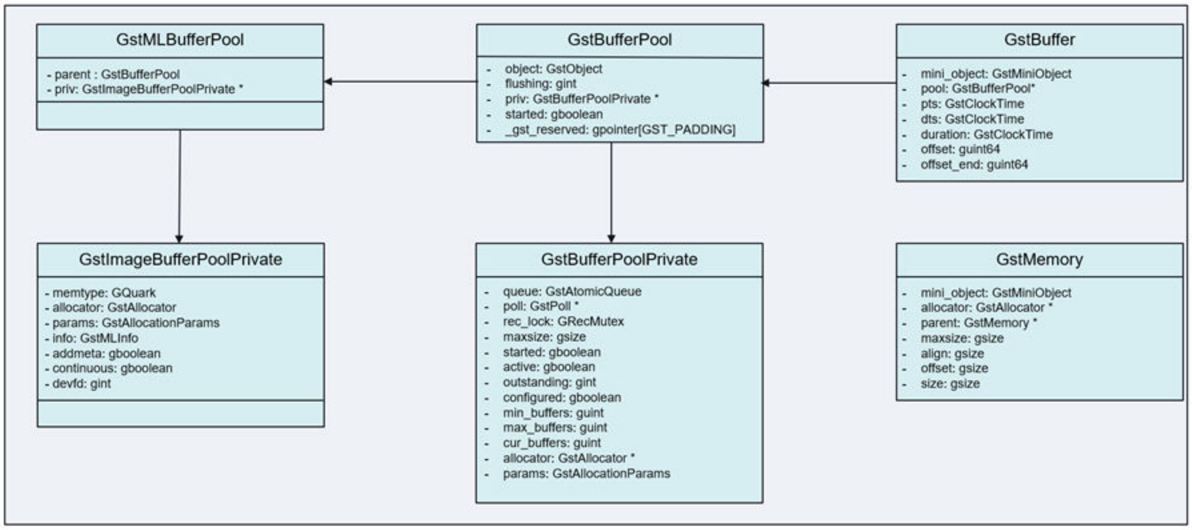


Figure 4-3 GstBufferPool subclass for ML

- `gst_ml_buffer_pool_new()`
    - **Description**

      Create a new bufferpool that can allocate ML frames

      ```
      GstBufferPool *
      gst_ml_buffer_pool_new (const gchar * type)
      ```
    - **Parameters**

      `type`–The memory that the pool will use for allocating buffers
    - **Returns**

      A new GstBufferPool to allocate ML frames, free with gst_object_unref.
- □ `GST_ML_BUFFER_POOL_TYPE_ION()`
    - **Description**

      A possible memory type that the pool will use for allocating buffers by passing it as an argument to gst_ml_buffer_pool_new.

      ```
      #define GST_ML_BUFFER_POOL_TYPE_ION "GstMLBufferPoolTypeIonMemory"
      ```

- □ `GST_ML_BUFFER_POOL_TYPE_SYSTEM()`
  - **Description**

    A possible memory type that the pool will use for allocating buffers by passing it as an argument to gst_ml_buffer_pool_new.

    ```
    #define GST_ML_BUFFER_POOL_TYPE_SYSTEM
    "GstMLBufferPoolTypeSystemMemory"
    ```

- □ `GST_ML_BUFFER_POOL_OPTION_TENSOR_META()`
  - **Description**

    An option that can be activated on bufferpool to request ML tensor metadata on buffers from the pool.

    ```
    #define GST_ML_BUFFER_POOL_OPTION_TENSOR_META
    "GstMLBufferPoolOptionTensorMeta"
    ```

- □ `GST_ML_BUFFER_POOL_OPTION_CONTINUOUS()`
  - **Description**

    An option that can be activated on bufferpool to request all tensors to be into a continuous physical memory.

    ```
    #define GST_ML_BUFFER_POOL_OPTION_CONTINUOUS
    "GstMLBufferPoolOptionContinuous"
    ```

4.3 Computer Vision APIs

- **GstCvMotionVector**

Table 4-3 Structure representing CV motion vector for a macro block

Field	Description
x (gint16)	Signed Horizontal motion vector for block 0
y (gint16)	Signed Vertical motion vector for block 0
confidence (gint8)	Motion vector confidence

- GstCvOptclFlowStats

Table 4-4 Structure representing CV optical flow statistics for a macro block

Field	Description
variance (guint16)	Macro block variance
mean (guint8)	Macro block mean
sad (guint16)	SAD (Sum of Absolute Differences) of the (0,0) motion vectors

- GstCvOptclFlowMeta

  Table 4-5 Extra buffer metadata structure describing CV optical flow properties

Field	Description
meta (GstMeta)	Parent GstMeta
id (guint)	ID corresponding to the memory index inside GstBuffer
mvectors (GstCvMotionVector)	Array containing motion vector data
stats (GstCvOptclFlowStats)	Array containing statistics for the motion vector data

APIs

- `gst_buffer_add_cv_optclflow_meta()`
  - **Description**

    Attaches GstCvOptclFlowMeta metadata to buffer with the given parameters.

    ```
    GstCvOptclFlowMeta *
    gst_buffer_add_cv_optclflow_meta (GstBuffer * buffer, GArray *
    mvectors, GArray * stats)
    ```

  - **Parameters**
    - `buffer` –A GstBuffer
    - `mvectors`–a array of GstCvMotionVector
    - `stats`–a array of GstCvOptclFlowStats
  - **Returns**

    The GstCvOptclFlowMeta on buffer. Don't free after the code is run.

- - `gst_buffer_get_cv_optclflow_meta()`
    - **Description**

      Get the first GstCvOptclFlowMeta on buffer.

      ```
      GstCvOptclFlowMeta *
      gst_buffer_get_cv_optclflow_meta (GstBuffer * buffer)
      ```

    - **Parameters**

      `buffer`–AGstBuffer
    - **Returns**

      The GstCvOptclFlowMeta on buffer. Don't free after the code is run.

- - `gst_buffer_get_cv_optclflow_meta_id()`
    - **Description**

      Find the GstCvOptclFlowMeta on buffer with the given ID.

      ```
      GstCvOptclFlowMeta *
      gst_buffer_get_cv_optclflow_meta_id (GstBuffer * buffer, guint id)
      ```

    - **Parameters**

      `buffer`–A GstBuffer

`id`–A metadata ID

- **Returns**

  The GstCvOptclFlowMeta on buffer. Don't free after the code is run.

5 Multimedia use cases

5.1 Camera

Prerequisites:

Run the following command:

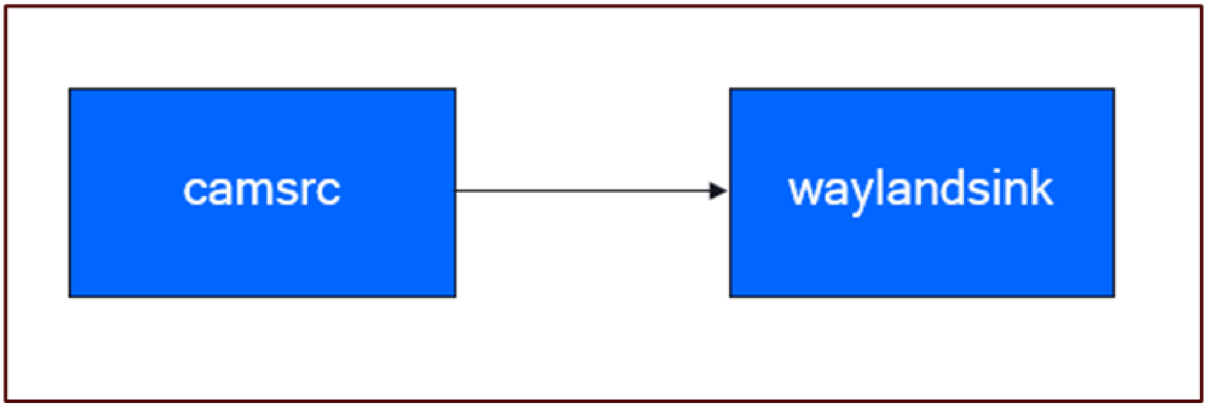
```
mount -o remount,rw /
mkdir -p /vendor/etc/camera
echo halBufferMgrMode=0 >/vendor/etc/camera/camxoverridesettings.txt
echo HALOutputBufferCombined=FALSE >> /vendor/etc/camera/
camxoverridesettings.txt
echo enableFeature2CTS=0 >> /vendor/etc/camera/camxoverridesettings.txt
```

If no IMU is attached, run this optional command:

```
echo enableNCSService=FALSE >> /vendor/etc/camera/camxoverridesettings.txt
export XDG_RUNTIME_DIR=/run/user/root
export WAYLAND_DISPLAY=wayland-1
```

5.1.1 Single 1080p YUV stream from live source

The pipeline demonstrates a single 1080p stream taken from camera and sent to display.



Run the following command to execute the pipeline:

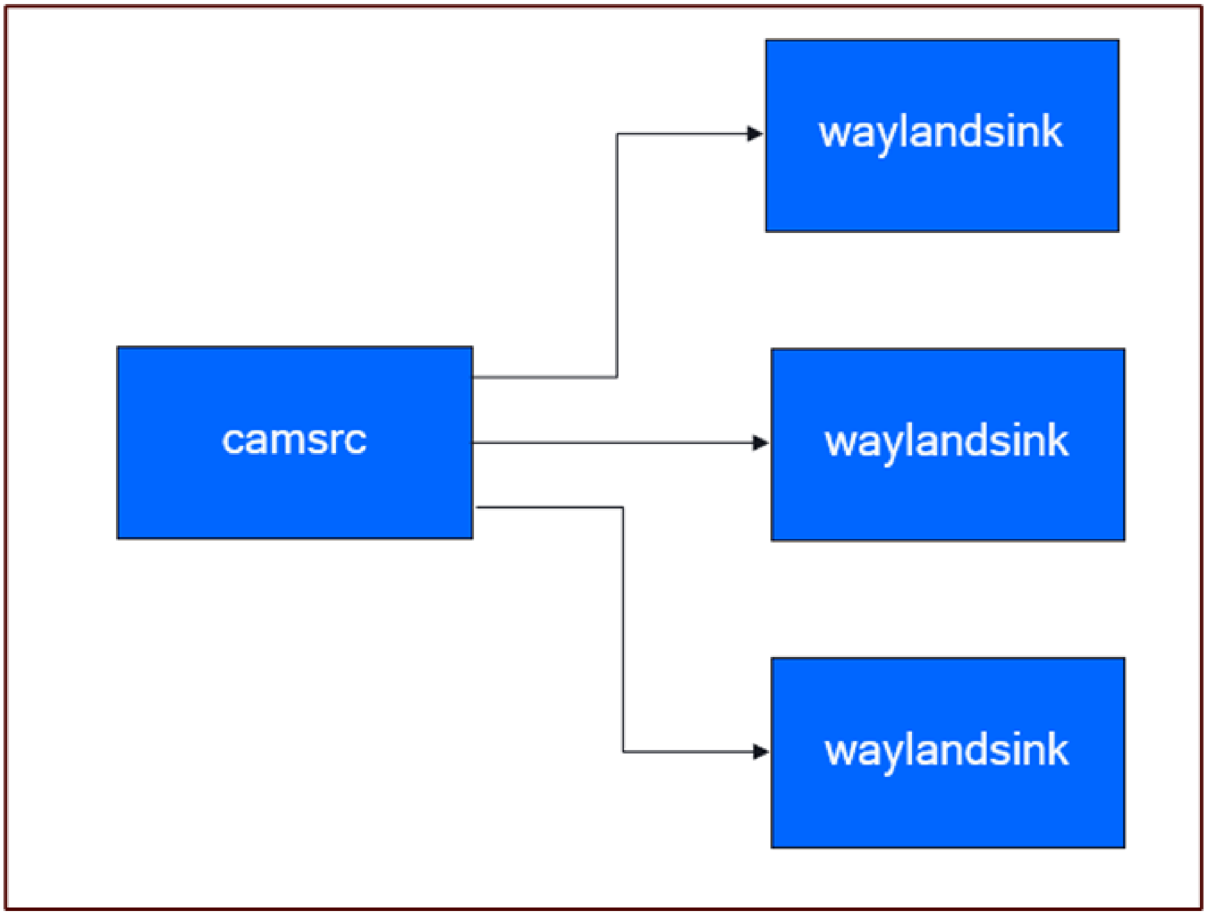
```
gst-launch-1.0 --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink
fullscreen=true async=true sync=false
```

To stop the use case, press **CTRL + C**.

The content from camera stream is displayed.

5.1.2 Three 1080p YUV streams from live source

The pipeline demonstrates three 1080p streams taken from camera and sent to display with each stream displayed at different positions on the screen.



Run the following command to execute the pipeline:

```
gst-launch-1.0 --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink x=0 y=0
width=500 height=400 async=true sync=false \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink x=510 y=0
width=500 height=400 async=true sync=false \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink x=0 y=410
width=500 height=400 async=true sync=false
```

To stop the use case, press **CTRL + C**.

The content from camera streams is displayed.

5.1.2.1 Use cases to interact with camera

This section provides the use cases on pipelines that use the GStreamer plug-in to interact with the camera.

main_preview_every_yuv_dump_4k30fps

Command with gst-launch-1.0:

```
export WAYLAND_DISPLAY=wayland-1
export XDG_RUNTIME_DIR=/run/user/root
GST_DEBUG=3 gst-launch-1.0 -e qtiqmmfsrc camera=0 name=camsrc ! video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink x=100
y=100 width=960 height=540 sync=false
```

Steps to execute the scenario:

1. Execute pre-condition
2. Execute command.
3. Check display or generated files

main_preview_1080P30fps_snapshot_1080p

Command with gst-launch-1.0:

```
gst-pipeline-app   -e qtiqmmfsrc name=camsrc !
                  video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1 ! multifilesink
                  enable-last-sample=false location=/data/output/frame%d.yuv
max-files=5
                  camsrc.image_1 ! "image/
jpeg,width=1920,height=1080,framerate=30/1"   !
                  multifilesink location=/data/frame%d.jpg sync=true
async=false
```

Steps to execute the scenario:

1. Execute pre-condition
2. Execute command.

   ```
   playing--p-- ( 5) camsrc-- (58) capture-image--0--Number of photos taken
   ```

3. Check display or generated files

main_preview_4k30fps_snapshot_4k

Command with gst-launch-1.0:

```
gst-pipeline-app   -e
                qtiqmmfsrc name=camsrc !
                video/x-
raw,format=NV12,width=3840,height=2160,framerate=30/1 ! multifilesink
                enable-last-sample=false location=/data/output/frame%d.yuv
max-files=5
                camsrc.image_1 ! "image/
jpeg,width=3840,height=2160,framerate=30/1"   !
                multifilesink location=/data/frame%d.jpg sync=true async=false
```

Steps to execute the scenario:

1. Execute pre-condition
2. Execute command:

   ```
   playing--p-- ( 5) camsrc-- (33) capture-image--0--Number of photos taken
   ```

3. Check display or generated files

main_preview_4k60fps_video_recording

Command with gst-launch-1.0:

```
gst-launch-1.0 -e qtiqmmfsrc camera=0 name=camsrc ! video/x-
raw,format=NV12,width=3840,height=2160,framerate=30/1 ! waylandsink x=100
y=100 width=960 height=540 sync=false camsrc. ! video/x-
raw,format=NV12,width=3840,height=2160,framerate=60/1 ! queue ! qtic2venc
control-rate=constant target-bitrate=10000000 ! queue ! h264parse ! mp4mux !
queue ! filesink location=/data/output/vid.mp4
```

main_video_recording_4k30fps_snapshot_4k

Command with gst-launch-1.0:

```
gst-pipeline-app   -e
                qtiqmmfsrc camera=0 name=camsrc !
                video/x-
raw,format=NV12,width=3840,height=2160,framerate=30/1 ! queue !   qtic2venc
                ! queue ! h264parse ! mp4mux ! queue ! filesink   location=/
data/vid.mp4
                camsrc.image_1 !   "image/jpeg,width=3840,height=2160" !
multifilesink
                location=/data/camera0%d.jpg   sync=true async=false
```

dual_camera_preview_every_yuv_dump_4k30fps

```
gst-launch-1.0   -e
                qtiqmmfsrc camera=0 name=camsrc !
                video/x-
raw,format=NV12,width=3840,height=2160,framerate=30/1 ! multifilesink
```

```
                enable-last-sample=false location=/data/client1_frame%d.yuv
max-files=5   qtiqmmfsrc
                camera=1 !   video/x-
raw,format=NV12,width=3840,height=2160,framerate=30/1 !
                multifilesink   enable-last-sample=false location=/data/
client2_frame%d.yuv
                max-files=5
```

single_camera_preview_1080p@30_fullscreen

1. Connect HDMI@1080p resolution.
2. Ensure that the Weston server is running.
3. Run the following command:

```
   export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 qtiqmmfsrc
                name=qmmf ! video/x-raw, format=NV12, width=1920,
height=1080,   framerate=30/1,
                camera=0 ! waylandsink fullscreen=true async=true
sync=false
```

single_camera_preview_1080p@30_fullscreen with GBM

1. Connect HDMI@1080p resolution.
2. Ensure that the Weston server is running.
3. Run the following command:

```
   export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0
                qtiqmmfsrc   name=qmmf ! video/x-raw\(memory:GBM\),
format=NV12, width=1920,
                height=1080,   framerate=30/1, camera=0 ! waylandsink
fullscreen=true async=true
                sync=false
```

single_camera_preview_4K@30_fullscreen

1. Connect HDMI@1080p resolution.
2. Ensure that the Weston server is running.
3. Run the following command:

```
   export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0
                qtiqmmfsrc   name=qmmf ! video/x-raw, format=NV12,
width=3840, height=2160,
                framerate=30/1, camera=0 ! waylandsink fullscreen=true
async=true
                sync=false
```

single_camera_preview_4K@30_fullscreen with GBM

1. Connect HDMI@1080p resolution.
2. Ensure that the Weston server is running.
3. Run the following command:

```
export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0
                qtiqmmfsrc   name=qmmf ! video/x-raw\(memory:GBM
\),format=NV12, width=3840,
                height=2160,   framerate=30/1, camera=0 ! waylandsink
fullscreen=true async=true
                sync=false
```

single_camera_preview_1080p@30_1/2screen

1. Connect HDMI@1080p resolution.
2. Ensure that the Weston server is running.
3. Run the following command:

```
export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0
                qtiqmmfsrc name=qmmf ! video/x-raw, format=NV12,
width=1920, height=1080,
                framerate=30/1, camera=0 ! waylandsink x=100 y=0 width=960
height=1080   async=true
                sync=false
```

single_camera_preview_720p@30_fullscreen

1. Connect HDMI@1080p resolution.
2. Ensure that the Weston server is running.
3. Run the following command:

```
export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0
                qtiqmmfsrc   camera=0 name=qmmf ! video/x-raw,
format=NV12, width=1280, height=720,
                framerate=30/1, camera=0 ! waylandsink fullscreen=true
async=true
                sync=false
```

parallel_2_camera_preview(2 camera)

1. Connect HDMI@1080p resolution.
2. Ensure that the Weston server is running.
3. Run the following command:

```
export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0
                qtiqmmfsrc   name=qmmf !
                "video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1" !   waylandsink
```

```
                sync=false x=0 y=0 width=600 height=600

               export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0
qtiqmmfsrc   camera=1 name=qmmf !
                video/x-raw, format=NV12, width=1280, height=720,
framerate=30/1 ! waylandsink
                x=650 y=0 width=600 height=600 async=true sync=false
```

5.1.2.2 Camera settings in Sensor mode

This section provides commands to execute the use cases that describe the camera settings in Sensor mode.

imx766_preview_1080p30fps_streaming_sensor_mode_0

gst-launch-1.0 command:

```
export
                XDG_RUNTIME_DIR=/run/user/root && gst-pipeline-app -e
qtiqmmfsrc camera=0
                name=camsrc sensor-mode=0 !
                video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1 ! multifilesink
                enable-last-sample=false location="/data/output/frame%d.yuv"
                max-files=3
```

Steps to execute the use case:

1. Run each sensor mode through gst settings and checking if it is working by logcat keyword: sensor mode
2. adb pull out the preview stream, the file can be view normally

NOTE If the YUV file is opened and the image is out of focus, choose focus_mode (continuous)

Observations:

```
adb logcat | grep "sensor mode"
I RecorderCameraContext: OpenCamera: Force sensor mode(0) received
 chxsensorselectmode.cpp:643 FindBestSensorMode() ***FORCING SENSOR MODE 0
-  for debug only
I CHIUSECASE: [CONFIG ] chxpipeline.cpp:384 CreateDescriptor()
Pipeline[Preview] Pipeline pointer 0x7fa53e1ca0 Selected sensor Mode W=9248,
H=6944
I CamX    : [CONFIG][CORE   ] camxsession.cpp:5915 SetRealtimePipeline()
Session 0x7fa566e3b0 Pipeline[Preview] Selected Sensor Mode W=9248  H=6944
```

imx766_preview_1080p30fps_streaming_sensor_mode_1

gst-launch-1.0 command:

```
export GST_PLUGIN_PATH=/usr/lib/gstreamer-1.0 && gst-launch-1.0 qtiqmmfsrc
camera=0 name=camsrc sensor-mode=1 ! video/x-
```

```
raw,format=NV12,width=1920,height=1080,framerate=30/1 ! multifilesink enable-
last-sample=false location="/data/output/frame%d.yuv" max-files=5
```

Steps to execute the use case:

1. Run each sensor mode via gst settings and checking if it is working by logcat keyword: sensor mode
2. adb pulls out the preview stream, the file can be view normally

Observations: The observations are same as the observations in imx766_preview_1080p30fps_streaming_sensor_mode_0.

imx766_preview_1080p30fps _streaming_sensor_mode_2

```
export GST_PLUGIN_PATH=/usr/lib/gstreamer-1.0 && gst-launch-1.0 qtiqmmfsrc
camera=0 name=camsrc sensor-mode=2 ! video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1 ! multifilesink enable-
last-sample=false location="/data/output/frame%d.yuv" max-files=5
```

Steps to execute the use case:

1. Run each sensor mode via gst settings and checking if it is working by logcat keyword: sensor mode
2. adb pull out the preview stream, the file can be view normally

Observations: The observations are same as the observations in imx766_preview_1080p30fps_streaming_sensor_mode_0.

imx766_preview_1080p30fps _streaming_sensor_mode_3

gst-launch-1.0 command:

```
export
                GST_PLUGIN_PATH=/usr/lib/gstreamer-1.0 && gst-launch-1.0
qtiqmmfsrc
                    camera=0 name=camsrc   sensor-mode=3 !
                video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1   ! multifilesink
                enable-last-sample=false   location="/data/output/frame%d.yuv"
                max-files=5
```

Steps to execute the use case:

1. Run each sensor mode via gst settings and checking if it is working by logcat keyword: sensor mode
2. adb pull out the preview stream, the file can be view normally

Observations: The observations are same as the observations in imx766_preview_1080p30fps_streaming_sensor_mode_0.

imx766_preview_1080p30fps _streaming_sensor_mode_4

gst-launch-1.0 command:

```
export
                GST_PLUGIN_PATH=/usr/lib/gstreamer-1.0 && gst-launch-1.0
qtiqmmfsrc
                    camera=0 name=camsrc   sensor-mode=4 !
                video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1   ! multifilesink
                enable-last-sample=false location="/data/output/frame%d.yuv"
                max-files=5
```

Steps to execute the use case:

1. Run each sensor mode via gst settings and checking if it is working by logcat keyword: sensor mode
2. adb pull out the preview stream, the file can be view normally

Observations: The observations are same as the observations in imx766_preview_1080p30fps_streaming_sensor_mode_0.

imx766_preview_1080p30fps _streaming_sensor_mode_5

gst-launch-1.0 command:

```
export
                GST_PLUGIN_PATH=/usr/lib/gstreamer-1.0 && gst-launch-1.0
qtiqmmfsrc
                    camera=0 name=camsrc   sensor-mode=5 !
                video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1   ! multifilesink
                enable-last-sample=false location="/data/output/frame%d.yuv"
                max-files=5
```

Steps to execute the use case:

1. Run each sensor mode via gst settings and checking if it is working by logcat keyword: sensor mode
2. adb pull out the preview stream, the file can be view normally

Observations: The observations are same as the observations in imx766_preview_1080p30fps_streaming_sensor_mode_0.

imx766_preview_1080p30fps _streaming_sensor_mode_6

gst-launch-1.0 command:

```
export
                GST_PLUGIN_PATH=/usr/lib/gstreamer-1.0 && gst-launch-1.0
qtiqmmfsrc
                    camera=0 name=camsrc   sensor-mode=6 !
                video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1   ! multifilesink
```

```
                enable-last-sample=false   location="/data/output/frame%d.yuv"
                max-files=5
```

Steps to execute the use case:

1. Run each sensor mode via gst settings and checking if it is working by logcat keyword: sensor mode
2. adb pull out the preview stream, the file can be view normally

Observations: The observations are same as the observations in imx766_preview_1080p30fps_streaming_sensor_mode_0.

imx766_preview_1080p30fps _streaming_sensor_mode_7

gst-launch-1.0 command:

```
export
                GST_PLUGIN_PATH=/usr/lib/gstreamer-1.0 && gst-launch-1.0
qtiqmmfsrc
                    camera=0 name=camsrc   sensor-mode=7 !
                video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1   ! multifilesink
                enable-last-sample=false location="/data/output/frame%d.yuv"
                max-files=5
```

Steps to execute the use case:

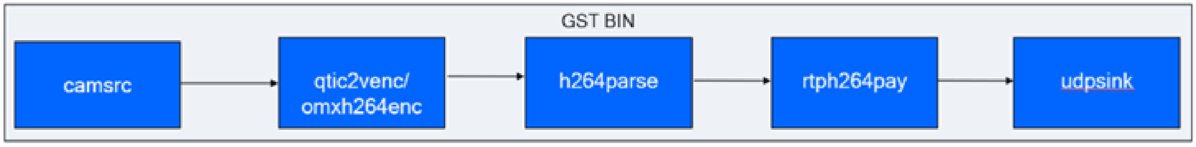
1. Run each sensor mode via gst settings and checking if it is working by logcat keyword: sensor mode
2. adb pull out the preview stream, the file can be view normally

Observations: The observations are same as the observations in imx766_preview_1080p30fps_streaming_sensor_mode_0.

5.2 Camera and video encode

5.2.1 One stream - 1080p AVC RTSP from live source

The pipeline demonstrates one 1080p stream taken from camera and sent to encoder with the stream encoded and sent over the network via rtsp streaming.



```
# Run RTSP server in a separate console on target with udpsrc (can be run in
backround as service):
gst-rtsp-server -p 8900 -m /live "( udpsrc name=pay0 port=8554 caps=
\"application/x-rtp,media=video,clock-rate=90000,encoding-
name=H264,payload=96\" )"
```

```
# Run the pipeline in same or another console on target:
gst-pipeline-app -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc
target-bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20 max-quant-
i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20 !
queue ! h264parse config-interval=-1 ! rtph264pay pt=96 ! udpsink
host=127.0.0.1 port=8554
```

To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

The streamed video is displayed as a RTSP stream on a host.

View RTSP stream on the host PC

Prerequisites:

- Install adb and VLC media player on the host machine.
- Set the binary/executable paths in the environment variables.
- Forward adb port by executing: adb forward tcp:8900 tcp:8900

For more information, contact QTI support.

On Linux host, do one of following:

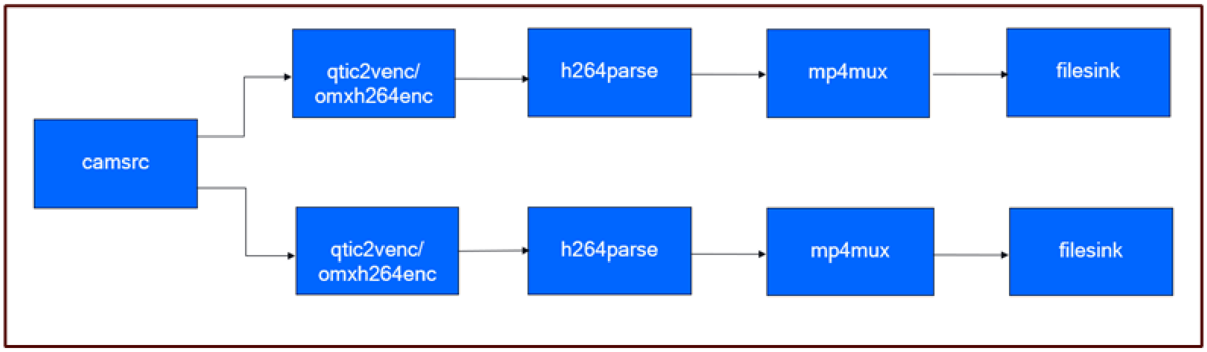
- vlc -vvv rtsp://127.0.0.1:8900/live [Ubuntu 18.04 with VLC version 3.0.8]
- ffplay -rtsp_transport tcp rtsp://127.0.0.1:8900/live

On Windows host, do the following:

1. Open VLC media player.
2. Go to **Media** > **Open Network Stream**, or press **CTRL + N**.
3. Enter rtsp://127.0.0.1:8900/live.
4. Click **Play**.

5.2.2 Two streams – 4K AVC and 480p AVC from live source

The pipeline demonstrates one 4k stream and one 1080p stream taken from camera and sent to encoder with each stream encoded and muxed into a different file.



Run the following command:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=3840,height=2160,framerate=30/1 ! queue ! qtic2venc
target-bitrate=6000000 ! queue ! h264parse ! mp4mux ! queue ! filesink
location="/data/mux1.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=640,height=480,framerate=30/1 ! queue ! qtic2venc target-
bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20 max-quant-i-
frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20 ! queue !
h264parse ! mp4mux ! queue ! filesink location="/data/mux2.mp4"
```

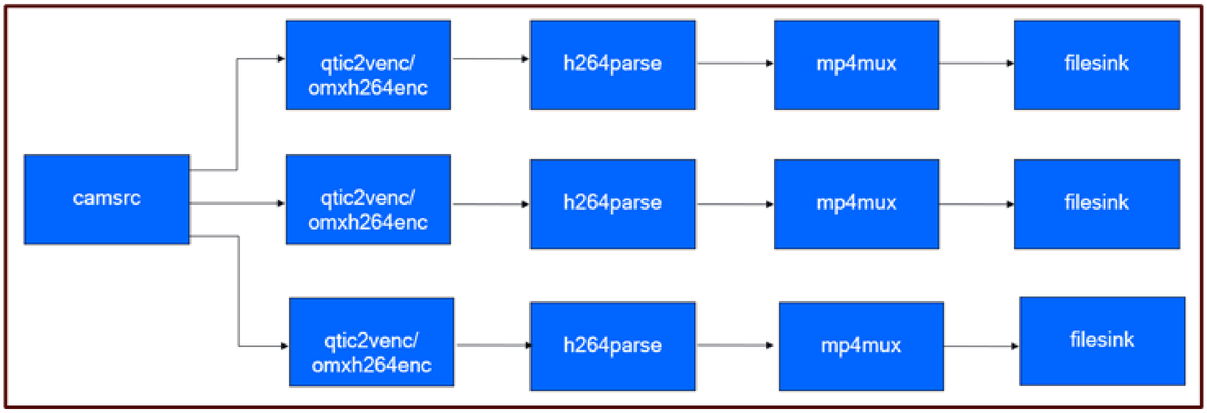
To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

The content from the pulled mp4 files should be playable.

```
adb pull /data/mux1.mp4
adb pull /data/mux2.mp4
```

5.2.3 Three 1080p AVC streams from live source

The pipeline demonstrates three 1080p streams taken from camera and sent to encoder with each stream encoded and muxed into a different file.



Run the commands to execute the pipeline:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux1.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux2.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux3.mp4"
```

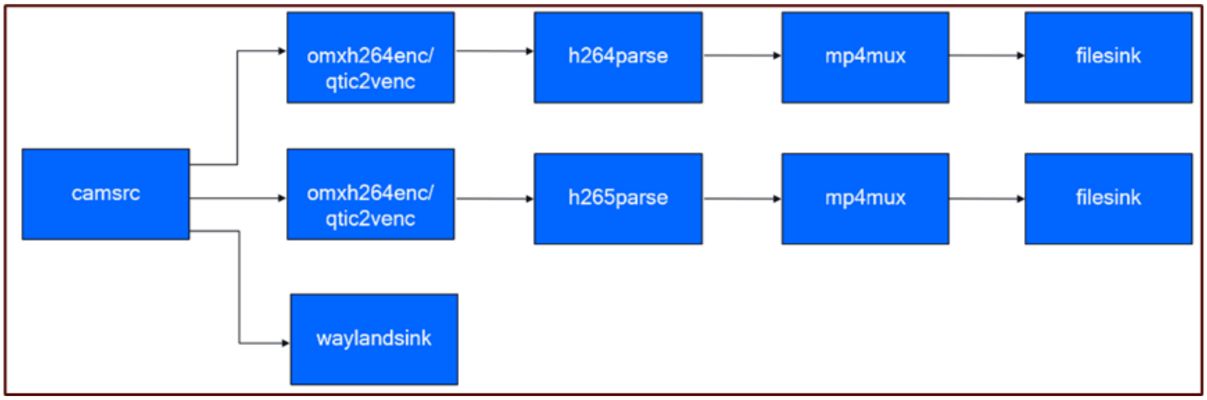
To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

The content from the pulled mp4 files should be playable.

```
adb pull /data/mux1.mp4
adb pull /data/mux2.mp4
adb pull /data/mux3.mp4
```

5.2.4 Three streams - 1080p AVC, 1080p HEVC, and 1080p YUV from live source

The pipeline demonstrates three 1080p stream taken from camera and two of them sent to encoder with one stream encoded as H264 and the other as HEVC and muxed into a different files. The third stream is sent to display.



Run the following commands to execute the pipeline:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000! queue !
h264parse ! mp4mux ! queue ! filesink location="/data/mux1.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc
target-bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20 max-quant-
i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20 !
queue ! h265parse ! mp4mux ! queue ! filesink location="/data/mux_hevc.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink sync=false
fullscreen=true enable-last-sample=false
```

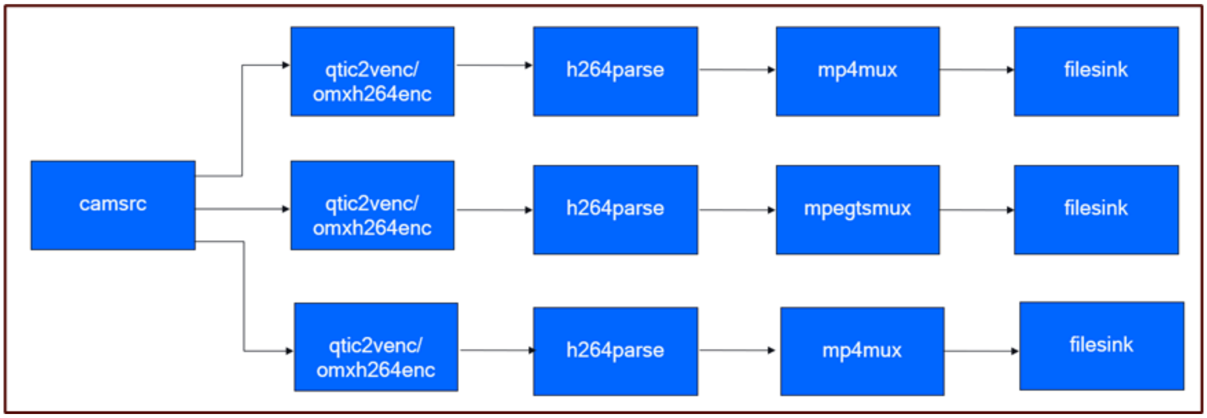
To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

The content from the pulled mp4 files should be playable and the stream from camera is displayed.

```
adb pull /data/mux1.mp4
adb pull /data/mux_hevc.mp4
```

5.2.5 Three streams - 1080p AVC MP4, 1080p AVC MPEGTS, and 1080p AVC MP4 from live source

The pipeline demonstrates three 1080p streams taken from camera and sent to encoder with each stream encoded as H264 and muxed - two as mp4 and one as mpegts into a different files.



Run the commands to execute the pipeline:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux1.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20  target-bitrate=6000000 !
queue ! h264parse ! mpegtsmux name=muxer ! queue ! filesink location="/data/
mux_mpehts.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux3.mp4"
```

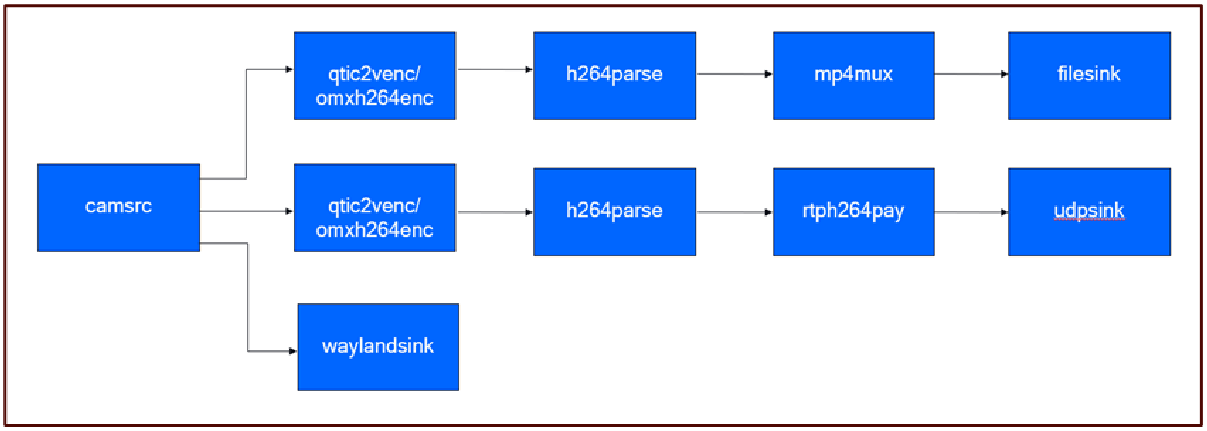
To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

The content from the pulled mp4 files should be playable.

```
adb pull /data/mux1.mp4
adb pull /data/mux_hevc.mp4
adb pull /data/mux3.mp4
```

5.2.6 Three streams – 1080p AVC file save , 1080p AVC RTSP, and 1080p YUV from live source

The pipeline demonstrates three 1080p streams taken from camera and two sent to encoder with each stream encoded as H264—one is MP4 muxed and save into a file while the other is sent over network through RTSP streaming. The third stream is sent to display.



Run the following command to execute the pipeline:

```
# Run RTSP server in a separate console on target with udpsrc (can be run in
backround as service):
gst-rtsp-server -p 8900 -m /live "( udpsrc name=pay0 port=8554 caps=
\"application/x-rtp,media=video,clock-rate=90000,encoding-
name=H264,payload=96\" )"

# Run the pipeline in same or another console on target:
gst-pipeline-app -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse config-interval=-1 ! rtph264pay pt=96 ! udpsink
host=127.0.0.1 port=8554 \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink
fullscreen=true async=true sync=false
```

To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

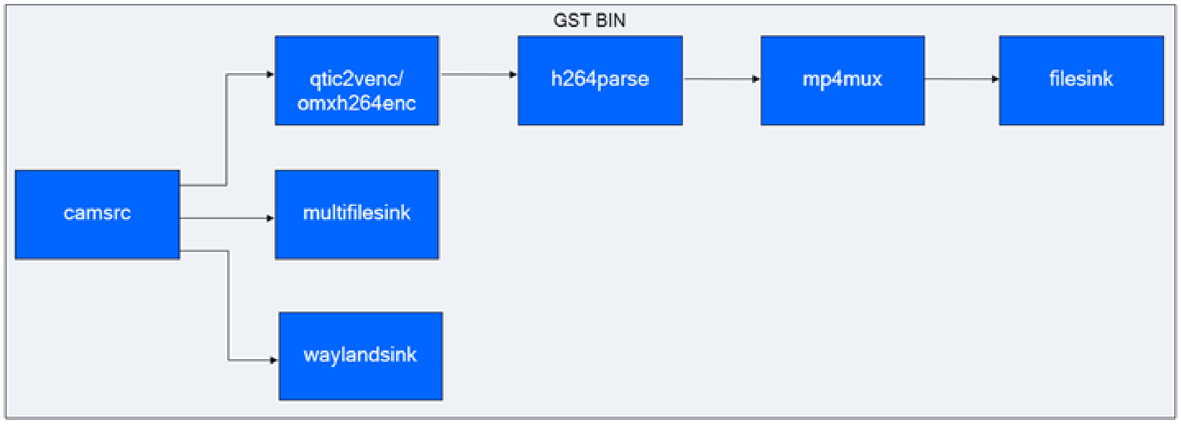
The content from the pulled mp4 file should be playable, the streamed video should be seen as a rtsp stream on a host and the stream from camera should be seen on display.

RELATED INFORMATION

“View RTSP stream on the host PC” on page 152

5.2.7 Three stream – 4K JPEG snapshot, 1080p AVC MP4, 1080p YUV from live source

The pipeline demonstrates three streams taken from camera—one 4k stream for JPEG snapshot, one 1080p stream, which is H264 encoded and MP4 muxed while one 1080p stream is sent to display.

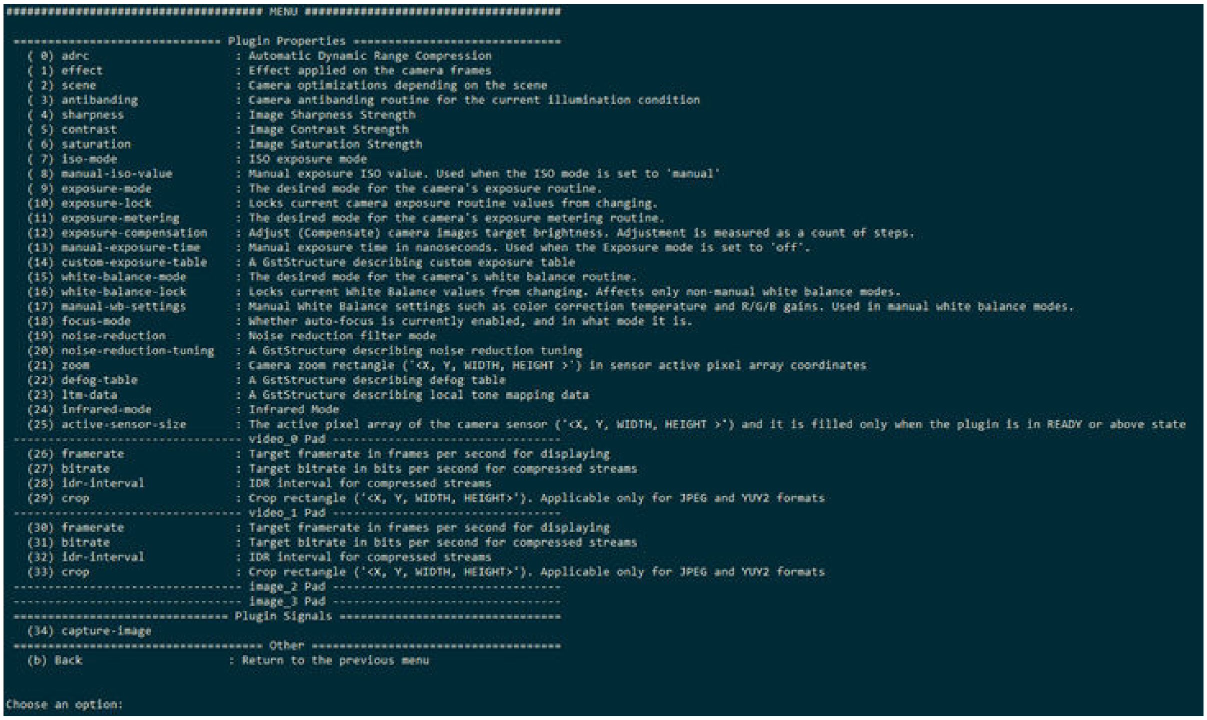


Run the commands to execute the pipeline:

```
gst-pipeline-app -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc
target-bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20 max-quant-
i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux.mp4" \
camsrc.image_1 ! "image/jpeg,width=3840,height=2160,framerate=30/1" !
multifilesink location=/data/frame%d.jpg sync=true async=false \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink
fullscreen=true async=true sync=false
```

A menu is displayed on the `cmd` line terminal. Enter name of the plug-in to be controlled: `camsrc`

The following options are displayed, select capture-image option to take snapshots:



```
##################################### MENU #####################################

 ============================== Plugin Properties ==============================
   ( 0) adrc                        : Automatic Dynamic Range Compression
   ( 1) effect                      : Effect applied on the camera frames
   ( 2) scene                       : Camera optimizations depending on the scene
   ( 3) antibanding                 : Camera antibanding routine for the current illumination condition
   ( 4) sharpness                   : Image Sharpness Strength
   ( 5) contrast                    : Image Contrast Strength
   ( 6) saturation                  : Image Saturation Strength
   ( 7) iso-mode                    : ISO exposure mode
   ( 8) manual-iso-value            : Manual exposure ISO value. Used when the ISO mode is set to 'manual'
   ( 9) exposure-mode               : The desired mode for the camera's exposure routine.
   (10) exposure-lock               : Locks current camera exposure routine values from changing.
   (11) exposure-metering           : The desired mode for the camera's exposure metering routine.
   (12) exposure-compensation       : Adjust (Compensate) camera images target brightness. Adjustment is measured as a count of steps.
   (13) manual-exposure-time        : Manual exposure time in nanoseconds. Used when the Exposure mode is set to 'off'.
   (14) custom-exposure-table       : A GstStructure describing custom exposure table
   (15) white-balance-mode          : The desired mode for the camera's white balance routine.
   (16) white-balance-lock          : Locks current White Balance values from changing. Affects only non-manual white balance modes.
   (17) manual-wb-settings          : Manual White Balance settings such as color correction temperature and R/G/B gains. Used in manual white balance modes.
   (18) focus-mode                  : Whether auto-focus is currently enabled, and in what mode it is.
   (19) noise-reduction             : Noise reduction filter mode
   (20) noise-reduction-tuning      : A GstStructure describing noise reduction tuning
   (21) zoom                        : Camera zoom rectangle ('<X, Y, WIDTH, HEIGHT >') in sensor active pixel array coordinates
   (22) defog-table                 : A GstStructure describing defog table
   (23) ltm-data                    : A GstStructure describing local tone mapping data
   (24) infrared-mode               : Infrared Mode
   (25) active-sensor-size          : The active pixel array of the camera sensor ('<X, Y, WIDTH, HEIGHT >') and it is filled only when the plugin is in READY or above state
 ---------------------------------- video_0 Pad ---------------------------------
   (26) framerate                   : Target framerate in frames per second for displaying
   (27) bitrate                     : Target bitrate in bits per second for compressed streams
   (28) idr-interval                : IDR interval for compressed streams
   (29) crop                        : Crop rectangle ('<X, Y, WIDTH, HEIGHT>'). Applicable only for JPEG and YUY2 formats
 ---------------------------------- video_1 Pad ---------------------------------
   (30) framerate                   : Target framerate in frames per second for displaying
   (31) bitrate                     : Target bitrate in bits per second for compressed streams
   (32) idr-interval                : IDR interval for compressed streams
   (33) crop                        : Crop rectangle ('<X, Y, WIDTH, HEIGHT>'). Applicable only for JPEG and YUY2 formats
 ---------------------------------- image_2 Pad ---------------------------------
 ---------------------------------- image_3 Pad ---------------------------------
 =============================== Plugin Signals ================================
   (34) capture-image
 ==================================== Other ====================================
   (b) Back                        : Return to the previous menu


Choose an option:
```

To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

The content from the pulled MP4 file and snapshots should be playable and the stream from camera is displayed:

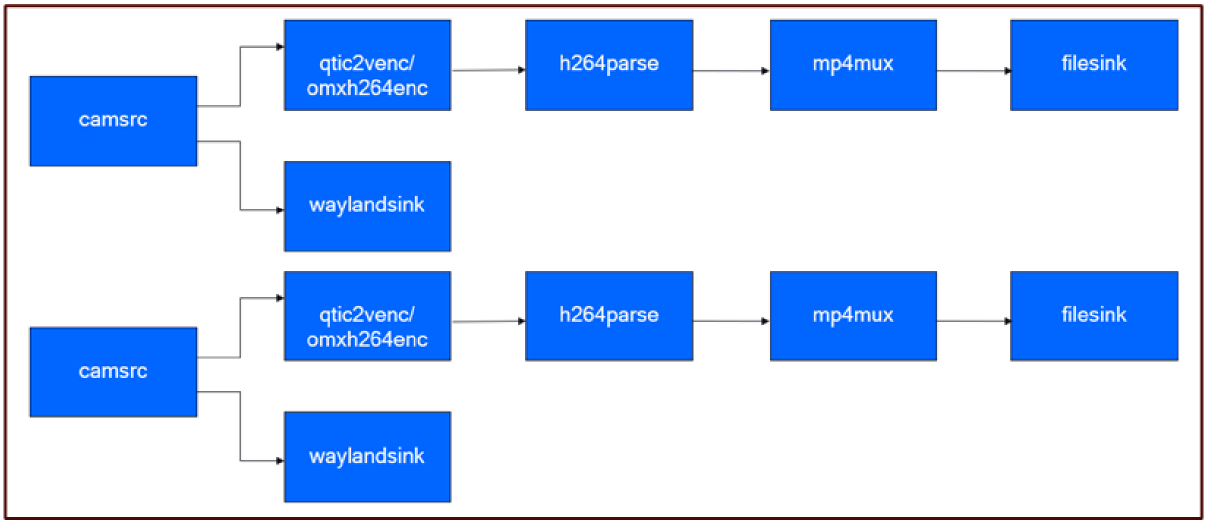
```
adb pull /data/mux.mp4
adb pull /data/frame*.jpg
```

5.3 Multi camera/Multi client use cases

5.3.1 Two streams each from main camera: 4k AVC MP4, 1080p YUV preview as Client 1 and secondary camera: 720p AVC, 720p YUV as Client2

The pipeline demonstrates two streams (4k and 1080p) taken from main camera with 4k stream sent to encoder with the stream encoded as H264 and muxed while 1080p is sent to display.

It also demonstrates two streams (both 720p) taken from secondary camera with one sent to encoder with the stream encoded as H264 and muxed while the other is sent to display.



Run the following command to execute the pipeline:

```
gst-launch-1.0 --gst-debug=2 \
qtiqmmfsrc name=camsrc_0 ! video/x-raw\(memory:GBM
\),format=NV12,width=3840,height=2160,framerate=30/1 ! queue ! qtic2venc
target-bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20 max-quant-
i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20  !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux1.mp4" \
camsrc_0. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink x=0 y=0
width=500 height=400 async=true sync=false \
qtiqmmfsrc name=camsrc_1 camera=1 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! queue ! qtic2venc min-quant-i-
frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-frames=30
quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 ! queue !
h264parse ! mp4mux ! queue ! filesink location="/data/mux2.mp4" \
camsrc_1. ! video/x-raw\(memory:GBM\), format=NV12, width=1280, height=720,
framerate=30/1 ! waylandsink x=510 y=0 width=500 height=400 async=true
sync=false
```

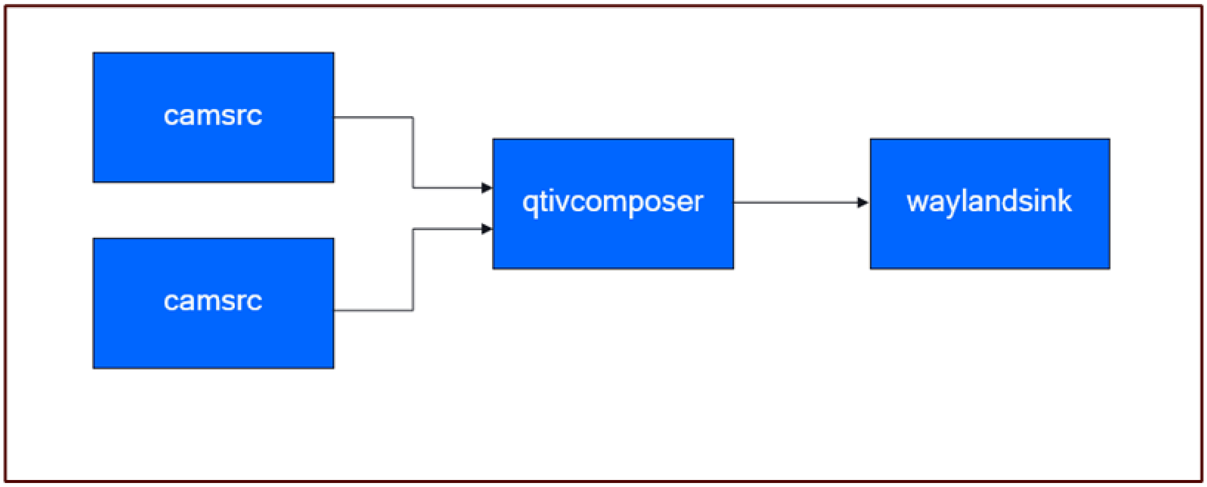
To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

The content from the pulled MP4 file and snapshots should be playable and the stream from camera are displayed.

```
adb pull /data/mux1.mp4
adb pull /data/mux2.mp4
```

5.3.2 Two streams – both 720p—one from each camera with side-by-side stitching sent to display

The pipeline demonstrates two streams (both 720p) taken one each from main camera and secondary camera with both sent to composer to be composed side by side and then displayed.



Run the commands to execute the pipeline:

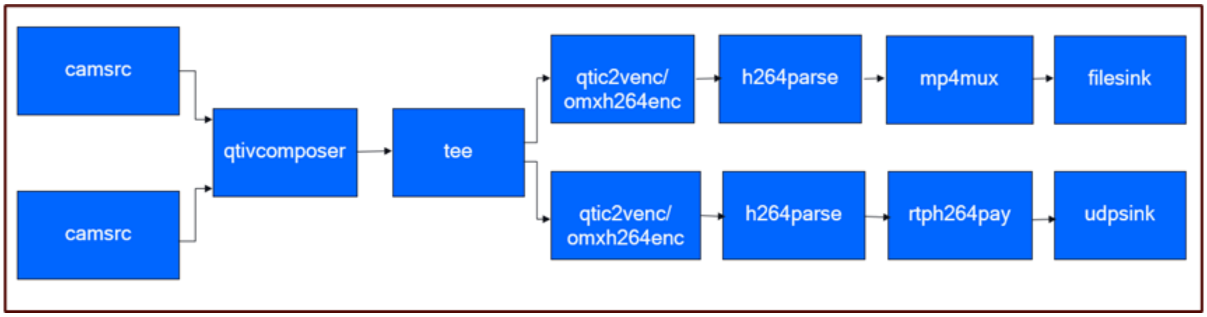
```
gst-launch-1.0 --gst-debug=2 \
qtivcomposer name=mixer sink_0::position="<0, 0>" sink_0::dimensions="<1280,
720>" \
sink_1::position="<0,  720>" sink_1::dimensions="<1280, 720>" \
mixer. ! queue ! waylandsink enable-last-sample=false async=true sync=false
fullscreen=true \
qtiqmmfsrc name=camsrc_0 camera=0 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! mixer. \
qtiqmmfsrc name=camsrc_1 camera=1 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! mixer.
```

To stop the use case, press **CTRL + C**.

The composed streams from camera are displayed.

5.3.3 Two streams – both 720p—one from each camera with side-by-side stitching sent to encode for filesave and RTSP streaming

The pipeline demonstrates two streams (both 720p) taken one each from main camera and secondary camera with both sent to composer to be composed side-by-side, and then duplicated with both encoded. However, one is muxed and saved in a file while the other is streamed via RTSP.



Run the commands to execute the pipeline:

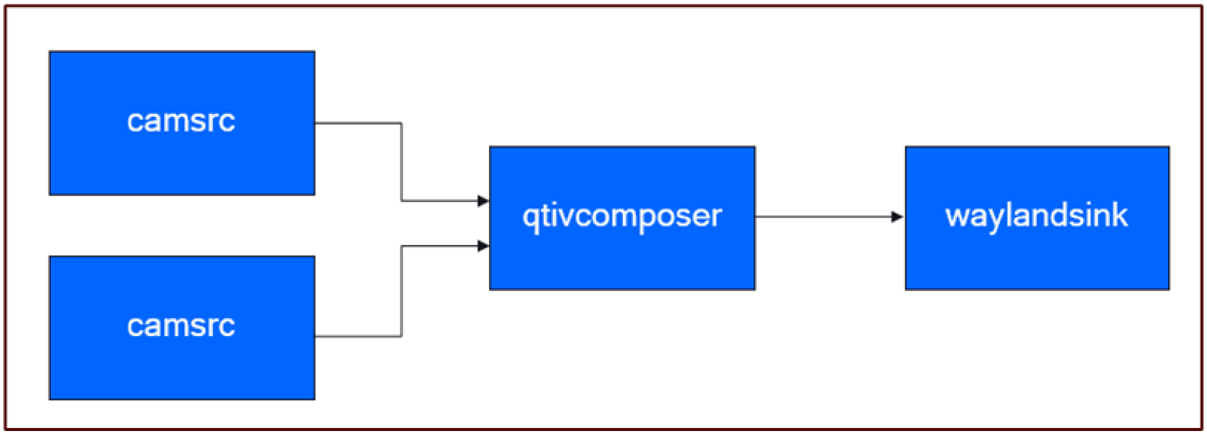
```
# Run RTSP server in a separate console on target with udpsrc (can be run in
backround as service):
gst-rtsp-server -p 8900 -m /live "( udpsrc name=pay0 port=8554 caps=
\"application/x-rtp,media=video,clock-rate=90000,encoding-
name=H264,payload=96\" )"

# Run the pipeline in same or another console on target:
gst-launch-1.0 --gst-debug=2 \
qtivcomposer name=mixer sink_0::position="<0, 0>" sink_0::dimensions="<1280,
720>" \
sink_1::position="<0,  720>" sink_1::dimensions="<1280, 720>" \
mixer. ! queue ! tee name=t_split \
t_split. ! queue ! qtic2venc target-bitrate=6000000 min-quant-i-frames=20 min-
quant-p-frames=20 max-quant-i-frames=30 max-quant-p-frames=30 quant-i-
frames=20 quant-p-frames=20! queue ! h264parse ! mp4mux ! queue ! filesink
location="/data/mux.mp4" \
t_split. ! queue ! qtic2venc target-bitrate=6000000 min-quant-i-frames=20 min-
quant-p-frames=20 max-quant-i-frames=30 max-quant-p-frames=30 quant-i-
frames=20 quant-p-frames=20 ! queue ! h264parse config-interval=-1 !
rtph264pay pt=96 ! udpsink host=127.0.0.1 port=8554 \
qtiqmmfsrc name=camsrc_0 camera=0 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! mixer. \
qtiqmmfsrc name=camsrc_1 camera=1 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! mixer.
```

To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

The content from the pulled mp4 file should be playable, the streamed video is displayed as RTSP stream on a host. See View RTSP stream on the host PC.

5.3.4 Two streams – both 720p—one from each camera with picture-in-picture composition and sent to display

The pipeline demonstrates two streams (both 720p) taken one each from main camera and secondary camera with both sent to composer to be composed as picture in picture and then sent to display.



Run the command to execute the pipeline:

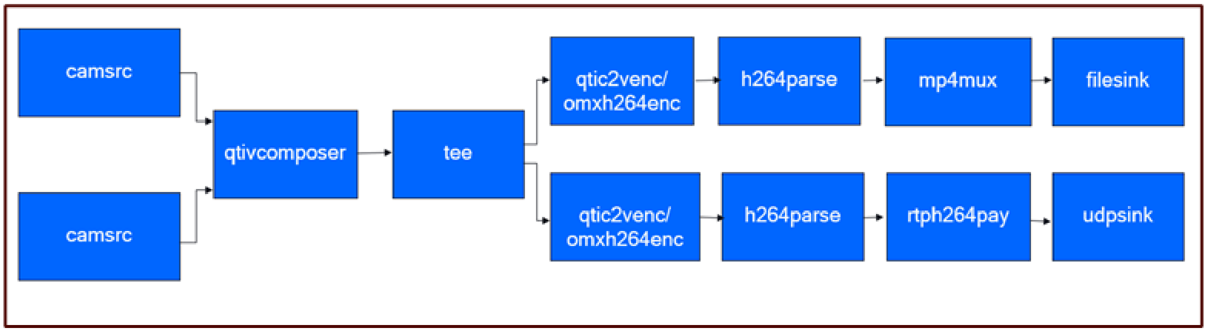
```
gst-launch-1.0 --gst-debug=2 \
qtivcomposer name=mixer sink_0::position="<0, 0>" sink_0::dimensions="<1280,
720>" \
sink_1::position="<590, 310>" sink_1::dimensions="<640, 360>" \
mixer. ! queue ! waylandsink enable-last-sample=false async=true sync=false
fullscreen=true \
qtiqmmfsrc name=camsrc_0 camera=0 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! mixer. \
qtiqmmfsrc name=camsrc_1 camera=1 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! mixer.
```

To stop the use case, press **CTRL + C**.

The composed streams from camera (picture in picture) are displayed.

5.3.5 Two streams – both 720p—one from each camera with picture-in-picture composition sent to encode for filesave and RTSP streaming

The pipeline demonstrates two streams (both 720p) taken one each from main camera and secondary camera with both sent to composer to be composed as picture in picture and then duplicated with both encoded but one is muxed and save in a file while the other is streamed via rtsp.



Run the commands to execute the pipeline:

```
# Run RTSP server in a separate console on target with udpsrc (can be run in
backround as service):
gst-rtsp-server -p 8900 -m /live "( udpsrc name=pay0 port=8554 caps=
\"application/x-rtp,media=video,clock-rate=90000,encoding-
name=H264,payload=96\" )"

# Run the pipeline in same or another console on target:
gst-launch-1.0 --gst-debug=2 \
qtivcomposer name=mixer sink_0::position="<0, 0>" sink_0::dimensions="<1280,
720>" \
sink_1::position="<590, 310>" sink_1::dimensions="<640, 360>" \
mixer. ! queue ! tee name=t_split \
t_split. ! queue ! qtic2venc target-bitrate=6000000 min-quant-i-frames=20 min-
quant-p-frames=20 max-quant-i-frames=30 max-quant-p-frames=30 quant-i-
frames=20 quant-p-frames=20 ! queue ! h264parse ! mp4mux ! queue ! filesink
location="/data/mux.mp4" \
t_split. ! queue ! qtic2venc target-bitrate=6000000 min-quant-i-frames=20 min-
quant-p-frames=20 max-quant-i-frames=30 max-quant-p-frames=30 quant-i-
frames=20 quant-p-frames=20 ! queue ! h264parse config-interval=-1 !
rtph264pay pt=96 ! udpsink host=127.0.0.1 port=8554 \
qtiqmmfsrc name=camsrc_0 camera=0 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! mixer. \
qtiqmmfsrc name=camsrc_1 camera=1 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! mixer.
```

To stop the use case, press **CTRL + C**, and then pull recorded content out from device using the following adb pull command, and play content on host PC.

The content from the pulled mp4 file should be playable, the streamed video is streamed as RTSP stream on a host. See View RTSP stream on the host PC.

5.3.6 1080p to 1080p – Rotate (90/180/270)

The pipeline demonstrates rotation of scenes from camera by 180 degree and display on local display device.

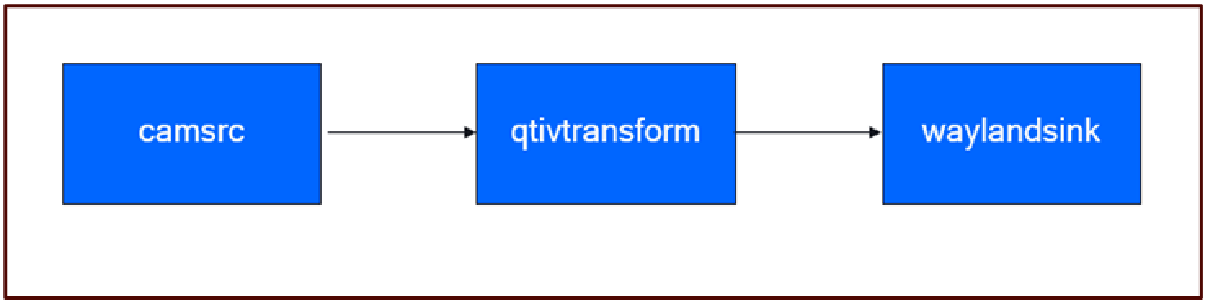


Figure 5-1 Rotation of scenes from camera by 180 degree

Run the command to execute the pipeline:

```
gst-launch-1.0 --gst-debug=2 qtiqmmfsrc name=camsrc camera=0 ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 !
qtivtransform rotate=180 ! waylandsink enable-last-sample=false async=true
sync=false fullscreen=true
```

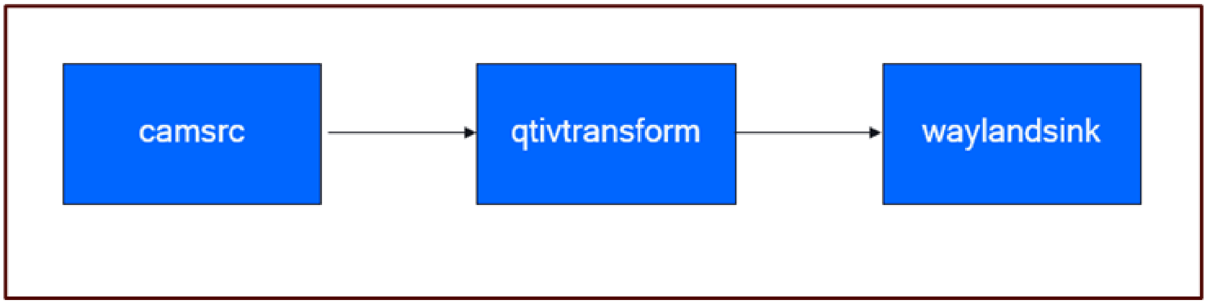
To stop the use case, press **CTRL + C**.

The transformed stream from camera is displayed.

5.4 Transform and Transcode use-cases

5.4.1 4K to 1080p – rotate (90/180/270) and downscale at same time

The pipeline demonstrates video stream of 4k resolution from camera being downscaled to 1080p and rotated by 180 degree and displayed to a local display device.



Run the command to execute the pipeline:

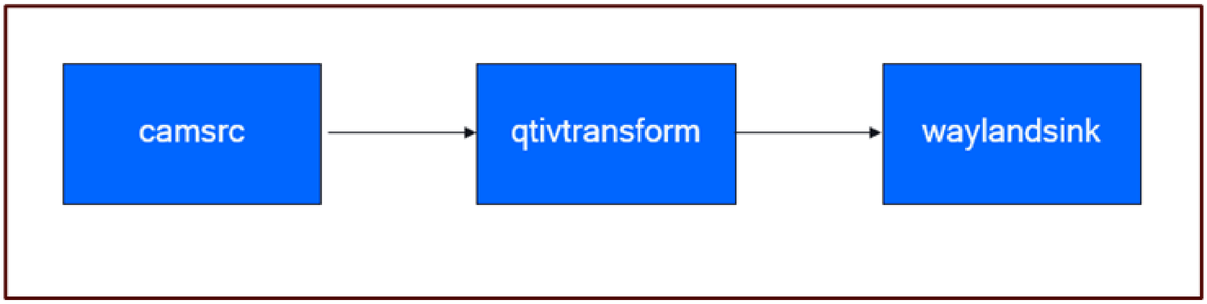
```
gst-launch-1.0 --gst-debug=2 qtiqmmfsrc name=camsrc camera=0 ! video/x-raw\
(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 !
qtivtransform rotate=180 ! video/x-raw\(memory:GBM\),width=1920,height=1080 !
waylandsink enable-last-sample=false async=true sync=false fullscreen=true
```

To stop the use case, press **CTRL + C**.

The transformed stream from camera is displayed.

5.4.2 1080p to 1080p – Flip Horizontal/Vertical flip

The pipeline demonstrates how to flip scenes horizontally.



Run the command to execute the pipeline:

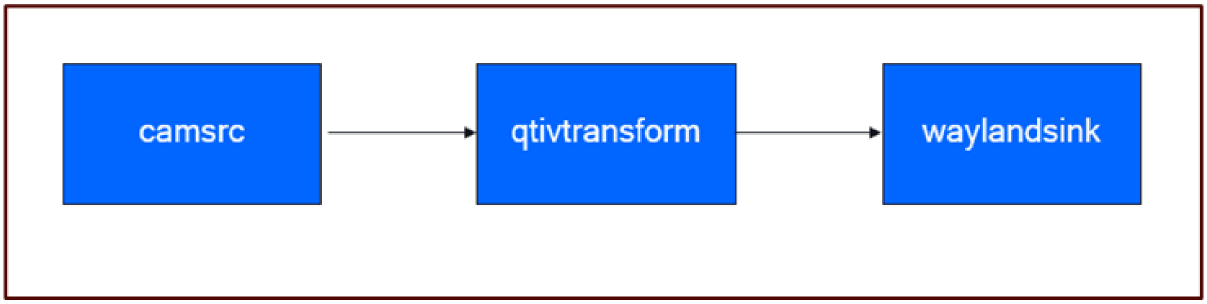
```
gst-launch-1.0 --gst-debug=2 qtiqmmfsrc name=camsrc camera=0 ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 !
qtivtransform flip-horizontal=true ! waylandsink enable-last-sample=false
async=true sync=false fullscreen=true
```

To stop the use case, press **CTRL + C**.

The transformed stream from camera is displayed.

5.4.3 4K to 1080p – Flip Horizontal/Vertical flip and Downscale at same time

The pipeline demonstrates how to downscale 4k resolution video stream to 1080p and flip the scenes horizontally.



Run the command to execute the pipeline:

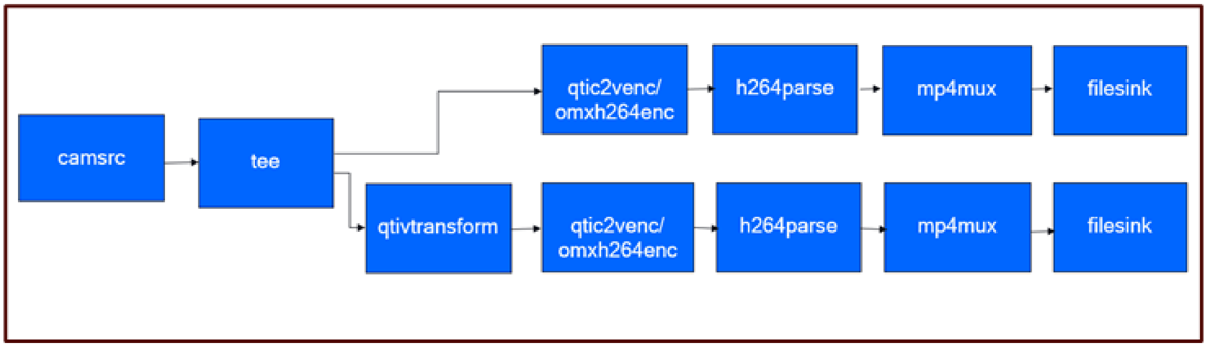
```
gst-launch-1.0 --gst-debug=2 qtiqmmfsrc name=camsrc camera=0 ! video/x-raw\
(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 !
qtivtransform flip-horizontal=true ! video/x-raw\(memory:GBM
\),width=1920,height=1080 ! waylandsink enable-last-sample=false async=true
sync=false fullscreen=true
```

To stop the use case, press **CTRL + C**.

The transformed stream from camera is displayed.

5.4.4 4K – Downscale to 1080p—encode to AVC and MP4 mux

The pipeline captures 4k resolution video stream, sends one copy of it to encoder and mp4 muxer to store in a file and sends another copy to qtivtransform to downscale the stream 1080p resolution before encoding and muxing to an mp4 file on device.



Run the comman to execute the pipeline:

```
gst-launch-1.0 --gst-debug=2 \
qtiqmmfsrc camera=0 ! video/x-raw\(memory:GBM
\),format=NV12,width=3840,height=2160,framerate=30/1 ! tee name=t_split \
t_split. ! queue ! qtic2venc min-quant-i-frames=20 min-quant-p-frames=20 max-
quant-i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20
target-bitrate=6000000 ! queue ! h264parse ! mp4mux ! queue ! filesink
location="/data/mux1.mp4" \
t_split. ! qtivtransform ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux2.mp4"
```

To stop the use case, press **CTRL + C**.

A few more use cases:

Table 5-1 Use cases on 4K resolution video stream

Use case	gst-launch 1.0 command	Observations

4k@30 HEVC -> Decode -> 4k@30 AVC Encode	`gst-launch-1.0 filesrc location=/data/input/7680_4320_H265_30fps.mp4 ! qtdemux ! queue ! h265parse ! qtic2vdec ! qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/output/mux2_fs.mp4"`
	`## OK gst-launch-1.0 -e qtiqmmfsrc camera=0 ! video/x-raw\(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 ! qtic2venc ! queue ! h265parse ! qtic2vdec ! qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/output/mux2_cs.mp4"`
		1. Generate video files success
		2. Generate video filess resolution/fps/format/ quality are matched setting
		3. No App hang, no crash

4k@30 AVC -> Decode -> 4k@30 HEVC Encode	`gst-launch-1.0 filesrc location=/data/input/3840_2160_H264_30fps.mp4 ! qtdemux ! queue ! h264parse ! qtic2vdec ! qtic2venc ! queue ! h265parse ! mp4mux ! queue ! filesink location="/data/output/mux3_fs.mp4"`
	`gst-launch-1.0 -e qtiqmmfsrc camera=0 ! video/x-raw,format=NV12,width=3840,height=2160,framerate=30/1 ! qtic2venc ! queue ! h264parse ! qtic2vdec ! qtic2venc ! queue ! h265parse ! mp4mux ! queue ! filesink location="/data/output/mux3_cs.mp4"`
		1. Generate video files success
		2. Generate video filess resolution/fps/format/ quality are matched setting
		3. No App hang, no crash

Table 5-1 Use cases on 4K resolution video stream (cont.)

Use case	gst-launch 1.0 command	Observations
8K@30 AVC -> Decode -> rotate & downscale -> 4K@30 AVC	`gst-launch-1.0 filesrc location=/data/input/7680_4320_H264_30fps.mp4 ! qtdemux ! queue ! h264parse ! qtic2vdec ! qtivtransform rotate=180 ! video/x-raw\(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 ! qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/output/mux7.mp4"`	1. Generate video files success 2. Generate video filess resolution/fps/format/quality are matched setting 3. No App hang, no crash
8K@30 AVC -> Decode -> Flip & downscale -> 4K@30 AVC	`gst-launch-1.0 filesrc location=/data/input/7680_4320_H264_30fps.mp4 ! qtdemux ! queue ! h264parse ! qtic2vdec ! qtivtransform flip-horizontal=true ! video/x-raw\(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 ! qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/output/mux8.mp4"`	1. Generate video files success 2. Generate video filess resolution/fps/format/quality are matched setting 3. No App hang, no crash
Video_qmmf_Transform_rotate_counter_clockwise_3840 x 2160 @30fps	`gst-launch-1.0 -e qtiqmmfsrc camera=0 ! video/x-raw\(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 ! qtic2venc ! queue ! h264parse ! qtic2vdec ! qtivtransform rotate=90CCW ! video/x-raw\(memory:GBM\) ! qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/output/mux10.mp4"`	1. Generate video files success 2. Generate video filess resolution/fps/format/quality are matched setting 3. No App hang, no crash

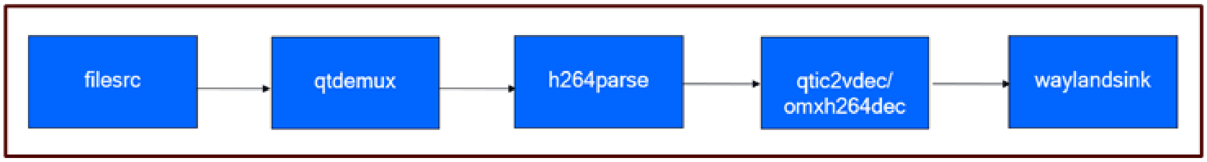
Table 5-1 Use cases on 4K resolution video stream (cont.)

Use case	gst-launch 1.0 command	Observations
Video_qmmf_Transform_scale_down_3840 x 2160 @30fps_to_1080p	`gst-launch-1.0 -e qtiqmmfsrc camera=0 ! video/x-raw\(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 ! qtic2venc ! queue ! h264parse ! qtic2vdec ! qtivtransform ! video/x-raw\(memory:GBM\),format=NV12,width=1920,height=1080 ! qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/output/mux12.mp4"`	1. Generate video files success 2. Generate video filess resolution/fps/format/quality are matched setting 3. No App hang, no crash
Video_qmmf_Transform_flip_horizontal_3840 x 2160 @30fps	`gst-launch-1.0 -e qtiqmmfsrc camera=0 ! video/x-raw\(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 ! qtic2venc ! queue ! h264parse ! qtic2vdec ! qtivtransform flip-horizontal=true ! video/x-raw\(memory:GBM\) ! qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/output/mux14.mp4"`	1. Generate video files success 2. Generate video filess resolution/fps/format/quality are matched setting 3. No App hang, no crash
Video_qmmf_Transform_flip_vertical_3840 x 2160 @30fps	`gst-launch-1.0 -e qtiqmmfsrc camera=0 ! video/x-raw\(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 ! qtic2venc ! queue ! h264parse ! qtic2vdec ! qtivtransform flip-vertical=true ! video/x-raw\(memory:GBM\) ! qtic2venc ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/output/mux15.mp4"`	1. Generate video files success 2. Generate video filess resolution/fps/format/quality are matched setting 3. No App hang, no crash

5.5 Video playback use cases

5.5.1 4K single stream video playback

The pipeline demonstrates playback of 4k Video stream from a file of media container format like mp4.



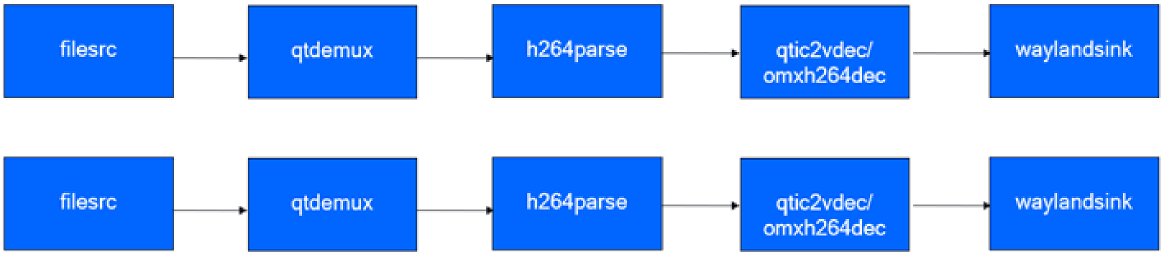
Run the command to execute the pipeline:

```
gst-launch-1.0 --gst-debug=2 filesrc location=/data/
Draw_2160p_180s_30FPS.mp4 ! qtdemux ! queue ! h264parse ! qtic2vdec !
waylandsink enable-last-sample=false async=false fullscreen=true
```

To stop the use case, press **CTRL + C**.

5.5.2 Two 1080p streams video playback simultaneously

The pipeline demonstrates playback of two video stream from a file of media container format like mp4 simultaneously.



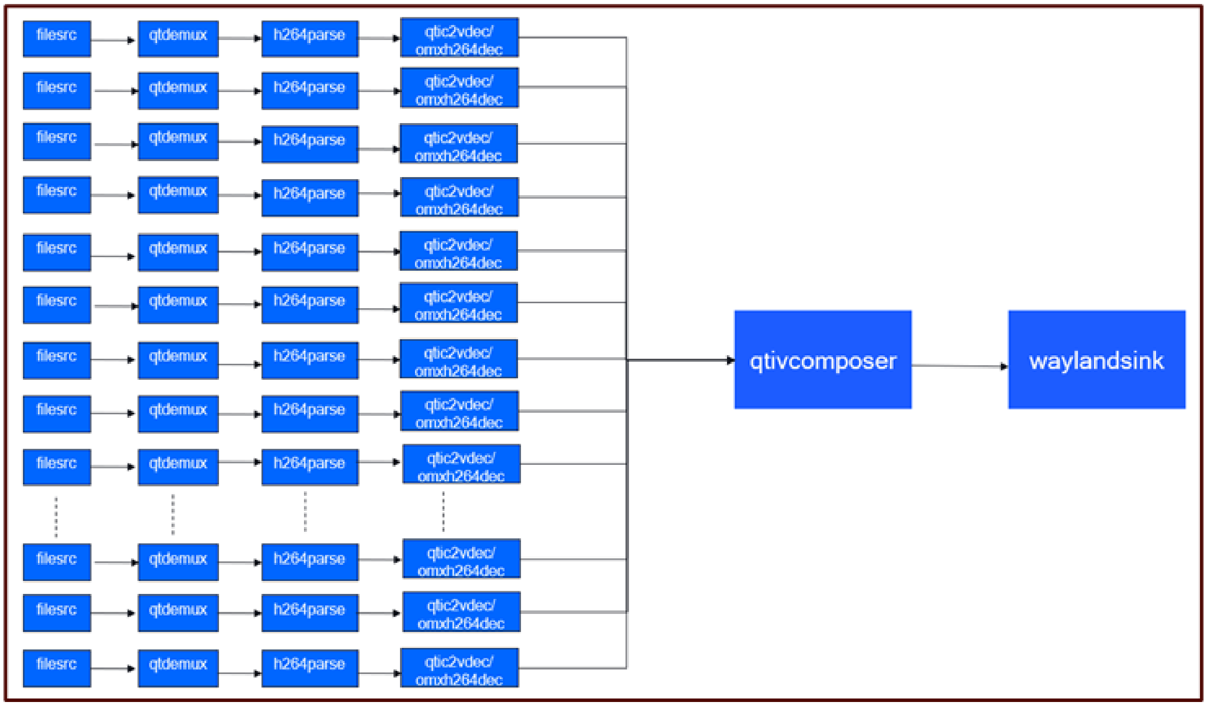
Run the command to execute the pipeline:

```
#In console 1:
gst-launch-1.0 --gst-debug=2 filesrc location=/data/
Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue ! h264parse ! qtic2vdec !
waylandsink enable-last-sample=false async=false fullscreen=true

#In Console 2:
gst-launch-1.0 --gst-debug=2 filesrc location=/data/
Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue ! h264parse ! qtic2vdec !
waylandsink enable-last-sample=false async=false fullscreen=true
```

To stop the use case, press **CTRL + C**.

5.5.3 16 1080p stream playback and side by side composition

The pipeline demonstrates use case about reading 16 1080p video streams from file sources and composing them together side by side and displaying them a single frame on local display device.



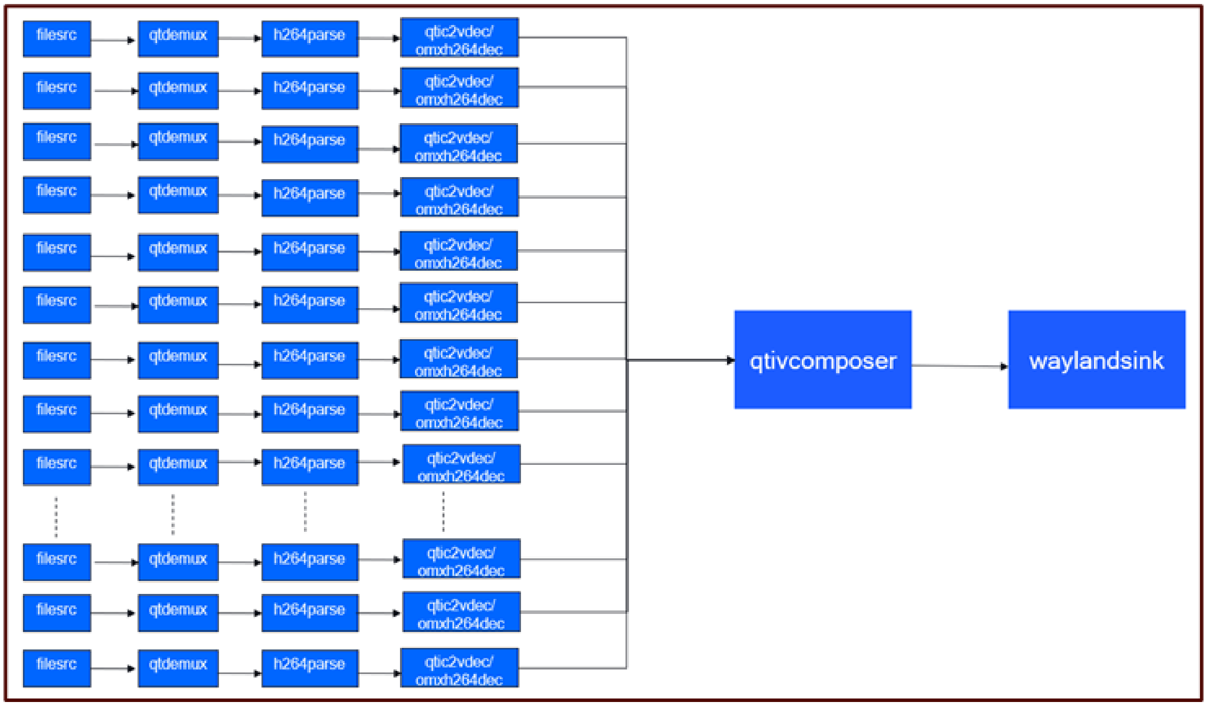
Run the commands to execute the pipeline:

```
gst-launch-1.0 --gst-debug=2 qtivcomposer name=mixer sink_0::position="<0,
0>" sink_0::dimensions="<480, 270>" \
sink_1::position="<480, 0>" sink_1::dimensions="<480, 270>" \
sink_2::position="<960, 0>" sink_2::dimensions="<480, 270>" \
sink_3::position="<1440, 0>" sink_3::dimensions="<480, 270>" \
sink_4::position="<0, 270>" sink_4::dimensions="<480, 270>" \
sink_5::position="<480, 270>" sink_5::dimensions="<480, 270>" \
sink_6::position="<960, 270>" sink_6::dimensions="<480, 270>" \
sink_7::position="<1440, 270>" sink_7::dimensions="<480, 270>" \
sink_8::position="<0, 540>" sink_8::dimensions="<480, 270>" \
sink_9::position="<480, 540>" sink_9::dimensions="<480, 270>" \
sink_10::position="<960, 540>" sink_10::dimensions="<480, 270>" \
sink_11::position="<1440, 540>" sink_11::dimensions="<480, 270>" \
sink_12::position="<0, 810>" sink_12::dimensions="<480, 270>" \
sink_13::position="<480, 810>" sink_13::dimensions="<480, 270>" \
sink_14::position="<960, 810>" sink_14::dimensions="<480, 270>" \
sink_15::position="<1440, 810>" sink_15::dimensions="<480, 270>" \
mixer. ! queue ! waylandsink enable-last-sample=false async=false
fullscreen=true \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer
```

To stop the use case, press **CTRL + C**.

5.5.4 24 720p/1080p stream video playback and side by side composition

The pipeline demonstrates use case about reading 24 720p video streams from file sources and composing them together side by side and displaying them a single frame on local display device.



Run the commands to execute the pipeline:

```
ulimit -n 4096 && gst-launch-1.0 --gst-debug=2 qtivcomposer name=mixer
sink_0::position="<0, 0>" sink_0::dimensions="<320, 180>" \
sink_1::position="<320, 0>" sink_1::dimensions="<320, 180>" \
sink_2::position="<640, 0>" sink_2::dimensions="<320, 180>" \
sink_3::position="<960, 0>" sink_3::dimensions="<320, 180>" \
sink_4::position="<1280, 0>" sink_4::dimensions="<320, 180>" \
sink_5::position="<1600, 0>" sink_5::dimensions="<320, 180>" \
sink_6::position="<0, 180>" sink_6::dimensions="<320, 180>" \
sink_7::position="<320, 180>" sink_7::dimensions="<320, 180>" \
sink_8::position="<640, 180>" sink_8::dimensions="<320, 180>" \
sink_9::position="<960, 180>" sink_9::dimensions="<320, 180>" \
sink_10::position="<1280, 180>" sink_10::dimensions="<320, 180>" \
sink_11::position="<1600, 180>" sink_11::dimensions="<320, 180>" \
sink_12::position="<0, 360>" sink_12::dimensions="<320, 180>" \
sink_13::position="<320, 360>" sink_13::dimensions="<320, 180>" \
sink_14::position="<640, 360>" sink_14::dimensions="<320, 180>" \
sink_15::position="<960, 360>" sink_15::dimensions="<320, 180>" \
sink_16::position="<1280, 360>" sink_16::dimensions="<320, 180>" \
sink_17::position="<1600, 360>" sink_17::dimensions="<320, 180>" \
sink_18::position="<0, 540>" sink_18::dimensions="<320, 180>" \
sink_19::position="<320, 540>" sink_19::dimensions="<320, 180>" \
sink_20::position="<640, 540>" sink_20::dimensions="<320, 180>" \
sink_21::position="<960, 540>" sink_21::dimensions="<320, 180>" \
sink_22::position="<1280, 540>" sink_22::dimensions="<320, 180>" \
sink_23::position="<1600, 540>" sink_23::dimensions="<320, 180>" \
mixer. ! queue ! waylandsink enable-last-sample=false async=false
fullscreen=true \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
```

```
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_720p_180s_24FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer.
```

To stop the use case, press **CTRL + C**.

Table 5-2 Use cases on video playback and display

Use case	Steps to execute the test case
single_video_playback_1080p 30_fullscreen	1. Connect HDMI 1080p resolution 2. Ensure that the Weston server is running 3. Run the command: `# export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/1920_1080_H265_30fps.mp4 ! qtdemux name=demux demux. ! queue ! h265parse ! qtic2vdec ! waylandsink fullscreen=true enable-last-sample=false`

single_video_playback_1080p 60_1/2screen	1. Connect HDMI 1080p resolution
	2. Ensure that the Weston server is running
	3. Run the command:
	`#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/qtc88_61.mp4 ! qtdemux name=demux demux. ! queue ! h264parse ! qtic2vdec ! waylandsink x=960 y=0 width=960 height=1080 enable-last-sample=false`
	4. Press **CTRL + C** to stop the playback and start new playback at a different location:
	`#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/qtc88_61.mp4 ! qtdemux name=demux demux. ! queue ! h264parse ! qtic2vdec ! waylandsink x=100 y=0 width=960 height=1080 enable-last-sample=false`

single_video_playback_4K 30_fullscreen	1. Connect HDMI 1080p resolution 2. Ensure that the Weston server is running 3. Run the command: `#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/QCOM_1_AAC_8_3840x2160_30fps_40Mbps.mp4 ! qtdemux name=demux demux. ! queue ! h264parse ! qtic2vdec ! waylandsink fullscreen=true enable-last-sample=false`
single_video_playback_1080p_1/4screen	1. Connect HDMI 1080p resolution 2. Ensure that the Weston server is running

Table 5-2 Use cases on video playback and display (cont.)

Use case	Steps to execute the test case

	3. Run the command:
	`#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/1920_1080_H265_30fps.mp4 ! qtdemux name=demux demux. ! queue ! h265parse ! qtic2vdec ! waylandsink x=0 y=0 width=960 height=540 enable-last-sample=false`
	4. Press **CTRL + C** to stop the playback and start new playback at a different location:
	`#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/1920_1080_H265_30fps.mp4 ! qtdemux name=demux demux. ! queue ! h265parse ! qtic2vdec ! waylandsink x=960 y=540 width=960 height=540 enable-last-sample=false`

single_video_playback_1080p_1/64screen	1. Connect HDMI 1080p resolution
	2. Ensure that the Weston server is running
	3. Run the command:
	`#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/1920_1080_H265_30fps.mp4 ! qtdemux name=demux demux. ! queue ! h265parse ! qtic2vdec ! waylandsink x=0 y=0 width=240 height=135 enable-last-sample=false`
	4. Press **CTRL + C** to stop the playback and start new playback at a different location:
	`#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/1920_1080_H265_30fps.mp4 ! qtdemux name=demux demux. ! queue ! h265parse ! qtic2vdec ! waylandsink x=960 y=540 width=240 height=135 enable-last-sample=false`

single_video_QVGA_playback_2x	1. Connect HDMI 1080p resolution 2. Ensure that the Weston server is running 3. Run the command: `#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/Qtc88_h264_320x240.mp4 ! qtdemux name=demux demux. ! queue ! h264parse ! qtic2vdec ! waylandsink x=0 y=0 width=640 height=240 enable-last-sample=false`
single_video_QVGA_playback_4x	1. Connect HDMI 1080p resolution 2. Ensure that the Weston server is running 3. Run the command: `#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/`

Table 5-2 Use cases on video playback and display (cont.)

Use case	Steps to execute the test case
	`Qtc88_h264_320x240.mp4 ! qtdemux name=demux demux. ! queue ! h264parse ! qtic2vdec ! waylandsink x=0 y=0 width=640 height=480 enable-last-sample=false`
single_video_QVGA_playback_16x	1. Connect HDMI 1080p resolution 2. Ensure that the Weston server is running 3. Run the command: `#export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 filesrc location=/data/Qtc88_h264_320x240.mp4 ! qtdemux name=demux demux. ! queue ! h264parse ! qtic2vdec ! waylandsink x=0 y=0 width=1280 height=960 enable-last-sample=false`

5.6 Audio use cases

5.6.1 Audio (only) capture

The single PCM capture pipeline demonstrates a single PCM stream taken from audio source and saved in a file. A single AAC capture pipeline demonstrates a single PCM stream taken from audio source, encoded as AAC and saved in a file

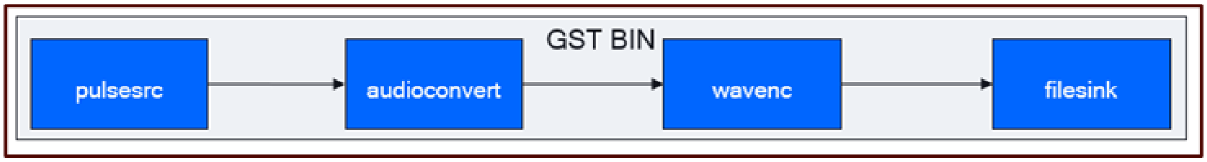


Figure 5-2 Single PCM stream

Run the command to execute the single PCM capture pipeline:

```
gst-launch-1.0 -v pulsesrc volume=10 ! audioconvert ! wavenc ! filesink
location=/data/Audio_PCM.wav
```

5.6.2 Audio (only) playback

The single AAC playback (compressed) offload pipeline demonstrates a single AAC stream taken from file source and played on speaker with decode and playback handled by DSP directly. There are two pipelines with a single MP3 stream, one where the stream is decoded and played on speaker, and another where the playback is handled by DSP directly.

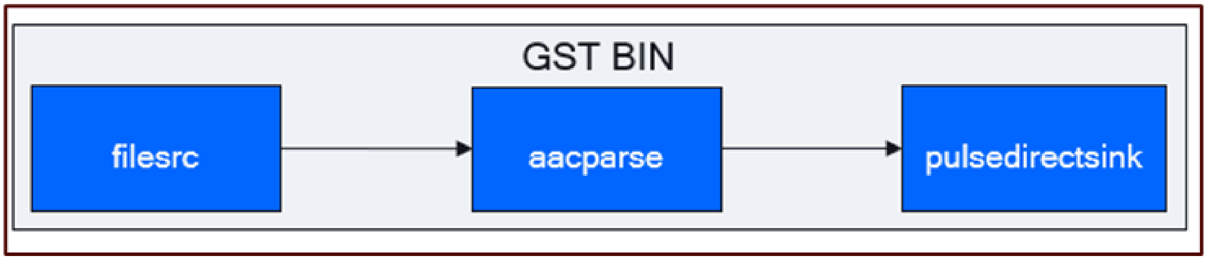


Figure 5-3 Single AAC stream taken from file source and played on speaker

Run the command to execute the single AAC playback (Compressed) offload pipeline:

```
gst-launch-1.0 filesrc location=<AAC file> ! aacparse ! pulsedirectsink
```

Use an AAC ADTS file in the AAC playback tests as they have audio meta per-frame.

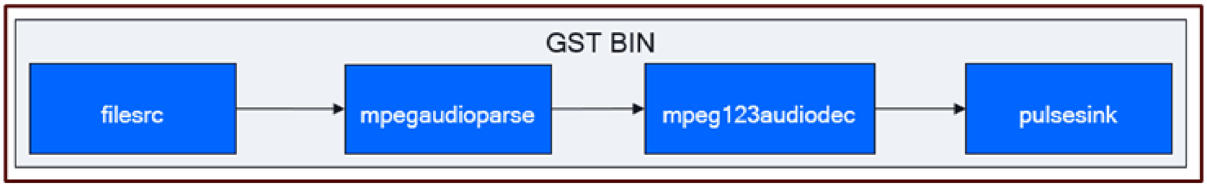


Figure 5-4 Single MP3 playback

Run the command to execute the single MP3 playback pipeline:

```
gst-launch-1.0 filesrc location=<MP3 file> ! mpegaudioparse !
mpg123audiodec ! pulsesink volume=10
```

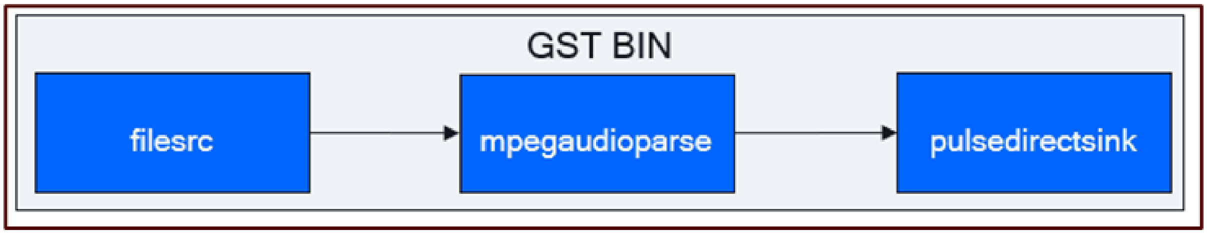


Figure 5-5 Single MP3 Playback (compressed offload)

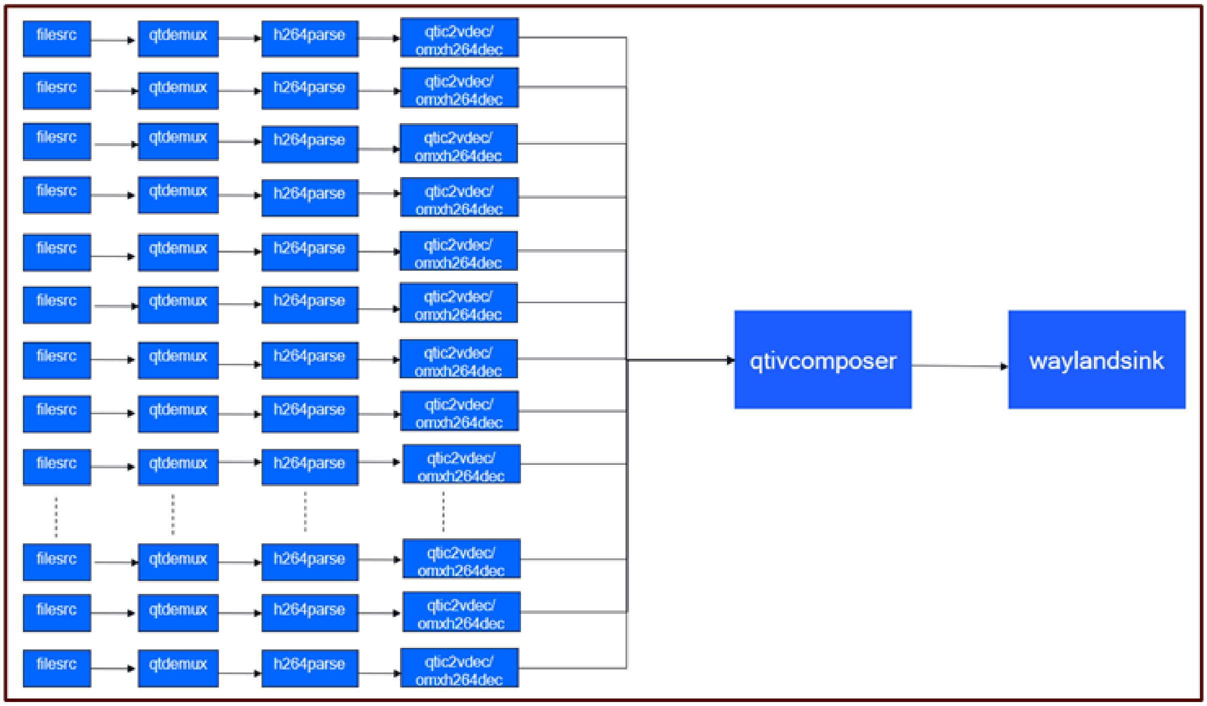
Run the command:

```
gst-launch-1.0 filesrc location=<MP3 file> ! mpegaudioparse ! pulsedirectsink
```

5.7 Video composition use cases

5.7.1 12 1080p offline stream side by side composition

The pipeline demonstrates use case about reading 12 1080p video streams from file sources and composing them together side by side and displaying them a single frame on local display device.

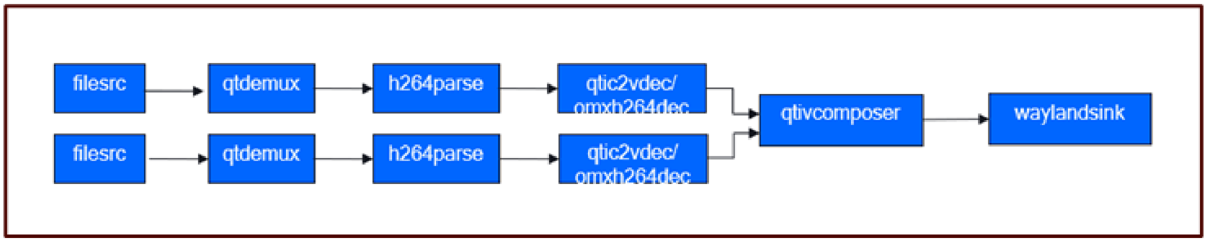


Run the commands to execute the pipeline. To stop the use case, press **CTRL + C**.

```
ulimit -n 4096 & gst-launch-1.0 --gst-debug=2 qtivcomposer name=mixer
sink_0::position="<0, 0>" sink_0::dimensions="<480, 270>" \
sink_1::position="<480, 0>" sink_1::dimensions="<480, 270>" \
sink_2::position="<960, 0>" sink_2::dimensions="<480, 270>" \
sink_3::position="<1440, 0>" sink_3::dimensions="<480, 270>" \
sink_4::position="<0, 270>" sink_4::dimensions="<480, 270>" \
sink_5::position="<480, 270>" sink_5::dimensions="<480, 270>" \
sink_6::position="<960, 270>" sink_6::dimensions="<480, 270>" \
sink_7::position="<1440, 270>" sink_7::dimensions="<480, 270>" \
sink_8::position="<0, 540>" sink_8::dimensions="<480, 270>" \
sink_9::position="<480, 540>" sink_9::dimensions="<480, 270>" \
sink_10::position="<960, 540>" sink_10::dimensions="<480, 270>" \
sink_11::position="<1440, 540>" sink_11::dimensions="<480, 270>" \
mixer. ! queue ! waylandsink enable-last-sample=false async=false
fullscreen=true \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer.
```

5.7.2 1080p + 1080p offline picture in picture composition

The pipeline demonstrates composition of two 1080p streams in a way to achieve picture in picture display. The two 1080p streams are read from two file sources.

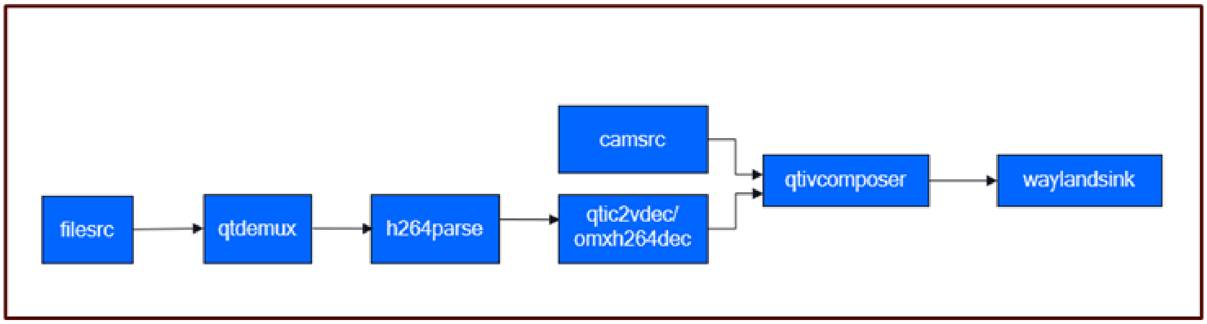


Run the commands to execute the pipeline. To stop the use case, press **CTRL + C**.

```
gst-launch-1.0 --gst-debug=2 \
qtivcomposer name=mixer sink_0::position="<0, 0>" sink_0::dimensions="<1280,
720>" \
sink_1::position="<590, 310>" sink_1::dimensions="<640, 360>" \
mixer. ! queue ! waylandsink enable-last-sample=false async=true sync=false
fullscreen=true \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer.
```

5.7.3 1080p live camera and 1080p offline picture in picture composition

The pipeline demonstrates composition of two 1080p streams in a way to achieve picture in picture display. One1080p stream is read from file source and second one is collected from camera.



Run the commands to execute the pipeline. To stop the use case, press **CTRL + C**.

```
gst-launch-1.0 --gst-debug=2 \
qtivcomposer name=mixer sink_0::position="<0, 0>" sink_0::dimensions="<1280,
720>" \
sink_1::position="<590, 310>" sink_1::dimensions="<640, 360>" \
mixer. ! queue ! waylandsink enable-last-sample=false async=true sync=false
fullscreen=true \
qtiqmmfsrc name=camsrc camera=0 ! video/x-raw, format=NV12, width=1920,
height=1080, framerate=30/1 ! mixer. \
filesrc location=/data/Draw_1080p_180s_30FPS.mp4 ! qtdemux ! queue !
h264parse ! qtic2vdec ! mixer.
```

Table 5-3 Use cases on video composition

Use case	gst-launch-1.0 command	Observations
8 1080p@30 side by side composition	`export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 --gst-debug=qtivcomposer:6 qtivcomposer name=mix sink_0::position="<0, 0>" sink_0::dimensions="<1280, 720>" sink_1::position="<1280, 0>" sink_1::dimensions="<1280, 720>" sink_2::position="<2560, 0>" sink_2::dimensions="<1280, 720>" sink_3::position="<0, 720>" sink_3::dimensions="<1280, 720>" sink_4::position="<2560, 720>" sink_4::dimensions="<1280, 720>" sink_5::position="<0, 1440>" sink_5::dimensions="<1280, 720>" sink_6::position="<1280, 1440>" sink_6::dimensions="<1280, 720>" sink_7::position="<2560, 1440>" sink_7::dimensions="<1280, 720>" mix. ! queue ! waylandsink sync=false fullscreen=true filesrc location=/data/input/1920_1080_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1920_1080_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1920_1080_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1920_1080_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1920_1080_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1920_1080_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1920_1080_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1920_1080_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix.`	Eight 1280x720 outputs arranged in a 3x3 grid displayed to local display.

Table 5-3 Use cases on video composition (cont.)

Use case	gst-launch-1.0 command	Observations
8 720p@30 side by side composition.	`export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 --gst-debug=qtivcomposer:6 qtivcomposer name=mix sink_0::position="<0, 0>" sink_0::dimensions="<1280, 720>" sink_1::position="<1280, 0>" sink_1::dimensions="<1280, 720>" sink_2::position="<2560, 0>" sink_2::dimensions="<1280, 720>" sink_3::position="<0, 720>" sink_3::dimensions="<1280, 720>" sink_4::position="<2560, 720>" sink_4::dimensions="<1280, 720>" sink_5::position="<0, 1440>" sink_5::dimensions="<1280, 720>" sink_6::position="<1280, 1440>" sink_6::dimensions="<1280, 720>" sink_7::position="<2560, 1440>" sink_7::dimensions="<1280, 720>" mix. ! queue ! waylandsink sync=false fullscreen=true filesrc location=/data/input/1280_720_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1280_720_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1280_720_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1280_720_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1280_720_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1280_720_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1280_720_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/1280_720_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix.`	Eight 1280x720 outputs arranged in a 3x3 grid displayed to local display.

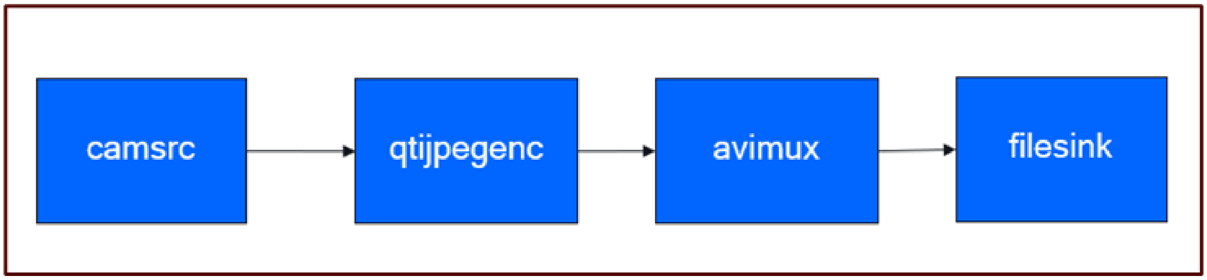
Table 5-3 Use cases on video composition (cont.)

Use case	gst-launch-1.0 command	Observations
4 4k@30 side by side composition	`export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 --gst-debug=qtivcomposer:6 qtivcomposer name=mix sink_0::position="<0, 0>" sink_0::dimensions="<1920, 1080>" sink_1::position="<1920, 0>" sink_1::dimensions="<1920, 1080>" sink_2::position="<0, 1080>" sink_2::dimensions="<1920, 1080>" sink_3::position="<1920, 1080>" sink_3::dimensions="<1920, 1080>" mix. ! queue ! waylandsink fullscreen=true filesrc location=/data/input/3840_2160_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/3840_2160_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/3840_2160_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix. filesrc location=/data/input/3840_2160_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! queue ! mix.`	Four 1920x1080 outputs arranged in a 2x2 grid displayed to local display.
Two 4K@30 picture in picture	`export XDG_RUNTIME_DIR=/run/user/root && gst-launch-1.0 --gst-debug=qtivcomposer:6 qtivcomposer name=mix sink_0::position="<0, 0>" sink_0::dimensions="<3840, 2160>" sink_1::position="<2560, 1440>" sink_1::dimensions="<1280, 720>" mix. ! queue ! waylandsink fullscreen=true filesrc location=/data/input/3840_2160_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! video/x-raw\(memory:GBM\) ! queue ! mix. filesrc location=/data/input/3840_2160_H264_30fps.mp4 ! qtdemux ! h264parse ! qtic2vdec ! video/x-raw\(memory:GBM\) ! queue ! mix.`	Two 4K recorded h264 videos. One is files played back on fullscreen while the other is displayed in the bottom right corner with resolution 1280x720

5.8 Camera and JPEG encode

5.8.1 Single stream JPEG encode and mux from live source

The pipeline demonstrates jpeg encode and muxing into an avi container.

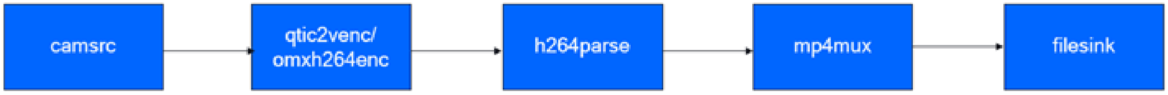


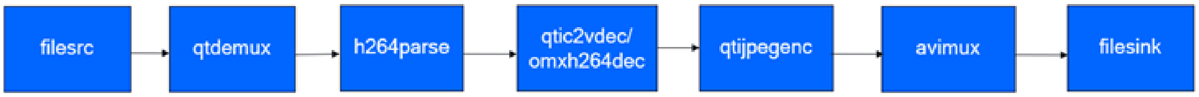
Run the commands to execute the pipeline. To stop the use case, press **CTRL + C**.

```
# Record H264 video stream:
gst-launch-1.0 -e qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=3840,height=2160,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux_4k.mp4"
```

5.8.2 Single stream JPEG Encode and mux from file source

The pipeline demonstrates jpeg encode and muxing into an avi container. The video stream used by JPEG encoder is obtained from file source.





Run the commands to execute the pipeline. To stop the use case, press **CTRL + C**.

```
# Record H264 video stream:
gst-launch-1.0 -e qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=3840,height=2160,framerate=30/1 ! queue ! qtic2venc min-
quant-i-frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 target-bitrate=6000000 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux_4k.mp4"
```

Use cases on camera and JPEG encode

- 4k Video (Preview) +JPEG : camera→ 4k jpeg snapshot

  This use case needs Weston to be running. The following are the commands to execute the use case.

  - Display

    ```
    export XDG_RUNTIME_DIR=/run/user/root && gst-pipeline-app -e qtiqmmfsrc
    name=camsrc camera=0 ! video/x-raw,format=NV12,width=3840,height=2160 !
    queue !  waylandsink fullscreen=true async=true camsrc.image_1 ! "image/
    jpeg,width=3840,height=2160" ! multifilesink location=/data/frame%d.jpg
    sync=true async=false
    ```

  - Encode

    ```
    export XDG_RUNTIME_DIR=/run/user/root && gst-pipeline-app -e qtiqmmfsrc
    name=camsrc camera=0 ! video/x-raw,format=NV12,width=3840,height=2160 !
    queue ! qtic2venc min-quant-i-frames=20 min-quant-p-frames=20 max-quant-
    i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-
    frames=20  ! queue ! h264parse ! mp4mux ! queue ! filesink location="/
    data/mux_4k_avc.mp4" camsrc.image_1 ! "image/
    jpeg,width=3840,height=2160" ! multifilesink location=/data/frame%d.jpg
    sync=true async=false
    ```

  Observations:

  The procedure was followed in the gst-pipeline-app:

  a. From the menu, choose the **PLAYING** option.

  b. Choose **Plugin Mode**.

  c. To capture snapshot, select **camsrc** plugin, and then select **capture-image**.

  d. Click **Back** to go back to the main menu and select **q** to quit.

  The JPEG snapshots are captured and verified. The display output is verified.

- 1080p Video (Preview) +RAW+JPEG : camera→ raw+ jpeg snapshot

  This use case needs Weston to be running. The following are the commands to execute the use case.

  - Display

    ```
    export XDG_RUNTIME_DIR=/run/user/root && gst-pipeline-app -e
    qtiqmmfsrc  name=camsrc camera=0  ! video/x-
    raw,format=NV12,width=1920,height=1080,framerate=30/1 ! queue !
    multifilesink enable-last-sample=false location="/data/output/yuv
    %d.yuv" max-files=5 camsrc.image_2 ! "image/
    jpeg,width=3840,height=2160" ! queue ! multifilesink location=/data/
    frame%d.jpg sync=true async=false  camsrc.image_3 ! "video/x-
    bayer,format=rggb,bpp=(string)10,width=4096,height=3072" !
    multifilesink location=/data/frame%d.raw sync=true async=false
    ```

    ```
    export XDG_RUNTIME_DIR=/run/user/root && gst-pipeline-app -e qtiqmmfsrc
    name=camsrc camera=0  ! video/x-
    raw,format=NV12,width=1920,height=1080 ! queue !  waylandsink
    ```

```
fullscreen=true async=true camsrc.image_1 !  "image/
jpeg,width=8192,height=6144" ! multifilesink location=/data/frame%d.jpg
sync=true async=false camsrc.image_2 ! "video/x-
bayer,format=rggb,bpp=(string)10,width=8192,height=6144" !
multifilesink location=/data/frame%d.raw sync=true async=false
```

- Encode

```
export XDG_RUNTIME_DIR=/run/user/root && gst-pipeline-app -e qtiqmmfsrc
name=camsrc camera=0  ! video/x-
raw,format=NV12,width=1920,height=1080,framerate=30/1 ! queue !
qtic2venc min-quant-i-frames=20 min-quant-p-frames=20 max-quant-i-
frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/
mux_1080_avc.mp4" camsrc.image_2 ! "image/
jpeg,width=3840,height=2160" ! queue ! multifilesink location=/data/
frame%d.jpg sync=true async=false  camsrc.image_3 ! "video/x-
bayer,format=rggb,bpp=(string)10,width=4096,height=3072" !
multifilesink location=/data/frame%d.raw sync=true async=false
```

```
export XDG_RUNTIME_DIR=/run/user/root && gst-pipeline-app -e qtiqmmfsrc
name=camsrc camera=0  ! video/x-
raw,format=NV12,width=1920,height=1080 ! queue ! qtic2venc min-quant-i-
frames=20 min-quant-p-frames=20 max-quant-i-frames=30 max-quant-p-
frames=30 quant-i-frames=20 quant-p-frames=20 ! queue ! h264parse !
mp4mux ! queue ! filesink location="/data/mux_1080_avc.mp4"
camsrc.image_1 !  "image/jpeg,width=8192,height=6144" ! multifilesink
location=/data/frame%d.jpg sync=true async=false camsrc.image_2 !
"video/x-bayer,format=rggb,bpp=(string)10,width=8192,height=6144" !
multifilesink location=/data/frame%d.raw sync=true async=false
```

Observations:

The procedure was followed in the gst-pipeline-app:

a. From the menu, choose the **PLAYING** option.

b. Choose **Plugin Mode**.

c. To capture snapshot, select **camsrc** plugin, and then select **capture-image**.

d. Click **Back** to go back to the main menu and select **q** to quit.

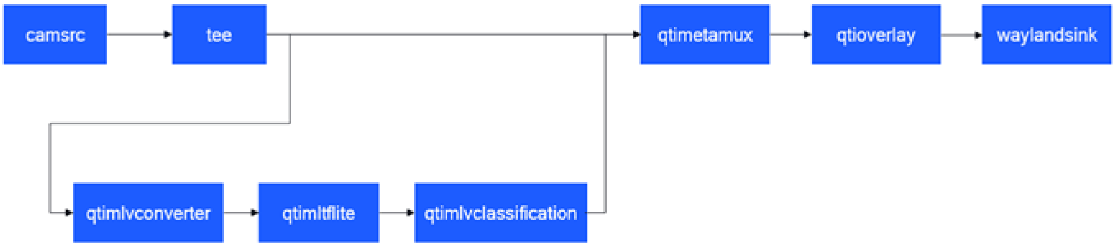
The JPEG and RAW snapshots are captured and verified. The display output is verified. The Chromatix tool was used to view the RAW snapshot.

6 Machine learning use cases

6.1 TensorFlow Lite use cases

6.1.1 Single camera stream with image classification and display

Variant 1: Use qtioverlay plugin to apply classification overlay



Use case is about using a mobilenet tflite model for classifying scenes from Video stream coming through camera source and overlaying classification labels using overlaylib and displaying the results to a local display.

1. The video stream is collected from camera source plugin and two copies are created - one is sent to qtimetamux plugin to retain the video stream and another is sent to ML inferencing pipeline
2. Our Preprocessing plugin, qtimlvconverter, receives the video stream on it's sink pad, does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input and finally converts the video stream to a tensor stream on it's source pad that mobilenet model can use for inferencing in later part of pipeline.
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, loads the mobilenet model, modifies the graph for the chosen delegate, receives the tensor stream on its sinkpad, executes the inference and produces tensor stream with the inference results on its source pad.

4. The postprocessing plugin for working on inference results from a classification model, qtimlvclassification converts the inference tensors it receives on its sinkpad into formats like video or text that our multimedia plugins can understand later.

   The qtimlvclassification plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of classification models.

   Here, in this use case, qtimlvclassification loads MobileNet submodule and produces results as structures of text and sends them to sinkpad of metamux.

5. The metamux plugin receives video stream and text stream with classification results corresponding to video stream on its sinkpads and produces gst buffers with contents of video stream on its sink pad, adding classification result from data sinkpad to gst buffer's meta (meta muxing) on its source pad.
6. The qtioverlay plugin receives the muxed stream and overlays the classification labels on the VideoFrame using CL and produces gst buffers with overlays in its source pad.
7. waylandsink submits the video stream its receiving on its sinkpad to weston and weston renders the video stream on a local display device.

See the video stream captured by camera source plugin and possible classifications generated for that scene on local display device.

Command:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
waylandsink sync=false fullscreen=true \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/mobilenet_v2_1.0_224_quant.tflite ! queue ! qtimlvclassification
threshold=60.0 results=3 module=mobilenet labels=/data/mobilenet.labels !
text/x-raw ! queue ! metamux.
```

To stop the use case, press **CTRL + C**.

Variant 2: Use qtivcomposer to mix original frame with classification mask

This use case is about using a MobileNet TFLite model for classifying scenes from video stream coming through camera source and composing classification labels and video stream together using qtivcomposer, and then displaying the results to a local display.

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to the qtivcomposer plugin to retain the video stream.
   - The other is sent to ML inferencing branch in the pipeline.
2. The preprocessing plugin, qtimlvconverter, receives the video stream on its sink pad.
   a. It performs preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input.
   b. It then converts the video stream to a tensor stream on its source pad that MobileNet model can use for inferencing in later part of pipeline.

3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite,does the following:
   a. Loads the mobilenet model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the inference results on its source pad
4. The qtimlvclassification postprocessing plugin works on inference results from a classification model. It is responsible for converting the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.
   - The qtimlvclassification plugin applies the threshold to the chosen number of results for the user.
   - This plugin can load corresponding modules for a variety of classification models.
   - Here, in this use case, qtimlvclassification loads MobileNet submodule and produces results as video frames with classification labels and sends them to sinkpad of qtivcomposer.
5. The qtivcomposer plugin receives original video stream and video stream with classification results on its sinkpads. It then produces on its sourcepad gst buffers with contents composed of video streams from its sinkpads.
6. Waylandsink submits the video stream it's receiving on its sinkpad to weston and weston renders the video stream on a local display device.

   See video stream captured by the camera source plugin and possible classifications generated for that scene on local display device.

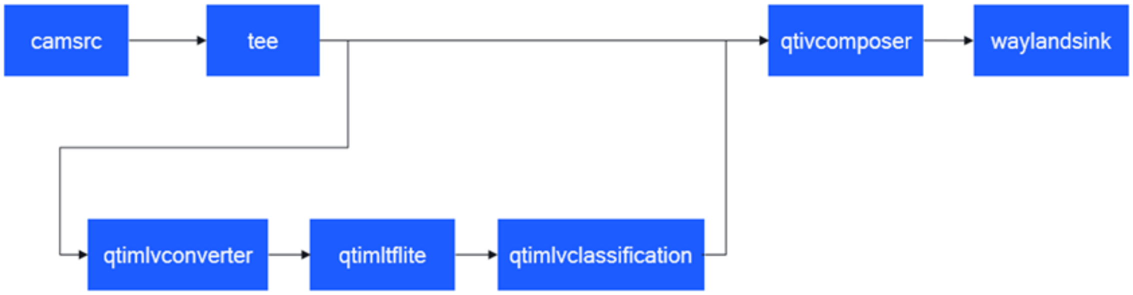


Figure 6-1 Single camera stream with Image Classification and Display

Commands:

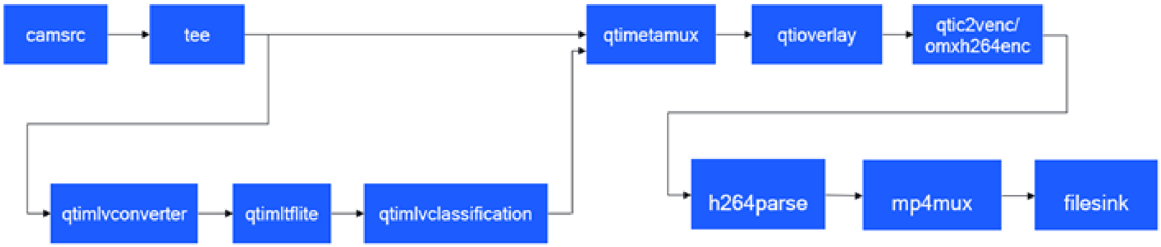
```
gst-launch-1.0 filesrc location=<MP3 file> ! mpegaudioparse !
mpg123audiodec ! pulsesink volume=10

gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::position="<50, 50>"
sink_1::dimensions="<368, 64>" ! queue ! waylandsink sync=false
```

```
fullscreen=true \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=gpu model=/
data/mobilenet_v2_1.0_224_quant.tflite ! queue ! qtimlvclassification
threshold=60.0 results=3 module=mobilenet labels=/data/mobilenet.labels !
video/x-raw,format=BGRA,width=368,height=64 ! queue ! mixer.
```

To stop the use case, press **CTRL + C**.

6.1.2 Single camera stream with image classification and encode

Variant 1: Use qtioverlay plugin to apply classification overlay



The use case is about using a MobileNet TFLite model for classifying scenes from video stream coming through camera source and overlaying classification labels using overlaylib and encoding this stream as h264 bitstream, later muxing in an mp4 container and finally storing it as an mp4 file.

1. The video stream is collected from camera source plugin and two copies are created - one is sent to qtimetamux plugin to retain the video stream and another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, receives the video stream on its sink pad, does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input and finally converts the video stream to a tensor stream on its source pad that mobilenet model can use for inferencing in later part of pipeline.
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, loads the MobileNet model, modifies the graph for the chosen delegate, receives the tensor stream on its sinkpad, executes the inference and produces tensor stream with the inference results on its source pad.
4. The postprocessing plugin for working on inference results from a classification model ,qtimlvclassification, converts the inference tensors it receives on its sinkpad into formats like video or text that our multimedia plugins can understand later.

   The qtimlvclassification plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of classification models.

   Here, in this use case, qtimlvclassification loads mobilenet submodule and produces results as structures of text and sends them to sinkpad of metamux.
5. The metamux plugin receives video stream and text stream with classification results corresponding to video stream on its sinkpads and produces gst buffers with contents of video stream on its sink pad, adding classification result from data sinkpad to gst buffer's meta (meta muxing) on its source pad.
6. The qtioverlay plugin receives the muxed stream and overlays the classification labels on the VideoFrame using CL and produces GST buffers with overlays in its source pad.

7. The qtic2venc pluin applies parameters to each frame of the video stream its receiving on its sinkpad and encodes it into bitstream and send it over its sourcepad.
8. h264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates containers format specification buffers
9. filesink stores the resulting stream in a file /data/video.mp4.
10. Pull video.mp4 from device and play it in a mediaplayer application.

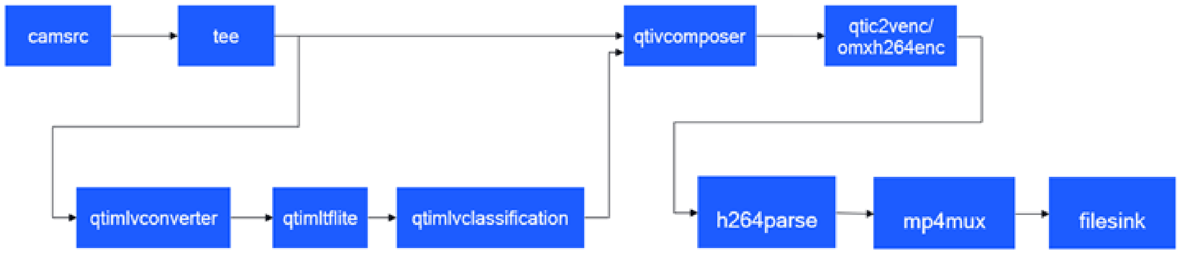
Command:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
qtic2venc target-bitrate=6000000 ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/mobilenet_v2_1.0_224_quant.tflite ! queue ! qtimlvclassification
threshold=60.0 results=3 module=mobilenet labels=/data/mobilenet.labels !
text/x-raw ! queue ! metamux
```

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
omxh264enc control-rate=max-bitrate target-bitrate=6000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/mobilenet_v2_1.0_224_quant.tflite ! queue ! qtimlvclassification
threshold=60.0 results=3 module=mobilenet labels=/data/mobilenet.labels !
text/x-raw ! queue ! metamux
```

To stop the use case, press **CTRL + C**.

Variant 2: Use qtivcomposer to mix original frame with classification mask



The use case is about using a MobileNet TFLite model to do the following:

1. Classify scenes from video stream coming through camera source
2. Compose the classification labels and video stream together using qtivcomposer

3. Encode this stream as h264 bitstream
4. Mux in an MP4 container and finally storing it as an mp4 file

The following is the sequence in which the use case is executed:

1. The video stream is collected from the camera source plugin and two copies are created:
   a. One is sent to the qtivcomposer plugin to retain the video stream.
   b. The other is sent to the ML inferencing branch in the pipeline.
2. The preprocessing plugin, qtimlvconverter does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that MobileNet model can use to inference in a later part of pipeline
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
   a. Loads the mobilenet model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the inference results on its source pad
4. The postprocessing plugin for working on inference results from a classification model ,qtimlvclassification converts the inference tensors it receives on its sinkpad into formats like video or text that our Multimedia plugins can understand later.

   The qtimlvclassification plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of classification models.

   Here, in this use case, qtimlvclassification loads MobileNet submodule and produces results as video frames with classification labels and sends them to sinkpad of qtivcomposer.
5. The qtivcomposer plugin receives original Video Stream and Video stream with classification results on it's sinkpads and produces on it's sourcepad gst buffers with contents composed of Video streams on it's sinkpads
6. The qtic2venc pluin applies parameters to each frame of the Video stream it's receiving on it's sinkpad and encodes it into bitstream and send it over it's sourcepad.
7. h264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates containers format specification buffers
8. filesink stores the resulting stream in a file /data/video.mp4.
9. Pull video.mp4 from device and play it in a media player application.

Command:

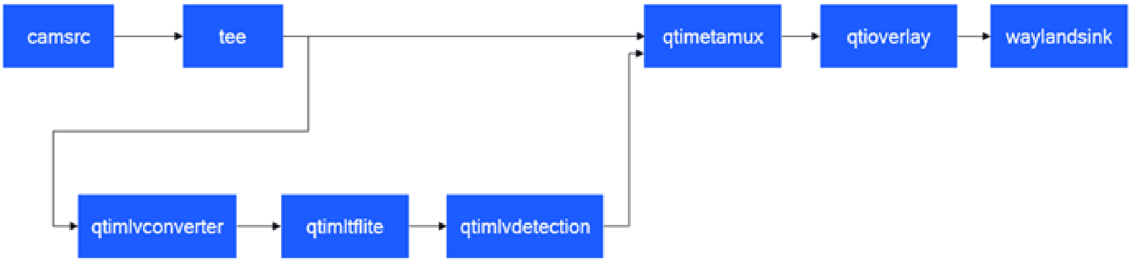
```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::position="<50, 50>"
sink_1::dimensions="<368, 64>" ! queue ! qtic2venc target-bitrate=6000000 !
h264parse ! queue ! mp4mux ! queue ! filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/mobilenet_v2_1.0_224_quant.tflite ! queue ! qtimlvclassification
threshold=60.0 results=3 module=mobilenet labels=/data/mobilenet.labels !
video/x-raw,format=BGRA,width=368,height=64 ! queue ! mixer.
```

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::position="<50, 50>"
sink_1::dimensions="<368, 64>" ! queue ! omxh264enc control-rate=max-bitrate
target-bitrate=6000000 interval-intraframes=29 periodicity-idr=1 !
h264parse ! queue ! mp4mux ! queue ! filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/mobilenet_v2_1.0_224_quant.tflite ! queue ! qtimlvclassification
threshold=60.0 results=3 module=mobilenet labels=/data/mobilenet.labels !
video/x-raw,format=BGRA,width=368,height=64 ! queue ! mixer.
```

To stop the use case, press **CTRL + C**.

6.1.3 Single camera stream with object detection and display

Variant 1: Use qtioverlay plugin to apply bounding box overlay



The use case is about using an yolov5m TFLite model to do the following:

1. Identify object in scene from video stream coming through camera source
2. Overlay bounding boxes over the detected objects using overlaylib
3. Display the results on a local display

The use case is executed in the following sequence:

1. The video stream is collected from camera source plugin and two copies are created:
   a. One is sent to the qtimetamux plugin to retain the video stream
   b. Another is sent to the ML inferencing pipeline

2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on it's source pad that yolov5m model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
   a. Loads the yolov5m model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the object detection results on its source pad
4. The postprocessing plugin for working on inference results from any object detection model, qtimlvdetection, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvdetection plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of detection models.

   Here, in this use case, qtimlvdetection loads yolov5m submodule and produces results as structures of text and sends them to sinkpad of metamux.
5. The metamux plugin receives video stream and text stream with bounding box results corresponding to video stream on its sinkpads and produces gst buffers with contents of video stream from its sink pad, adding bounding boxes as GstVideoRegionOfInterest from data sinkpad to gst buffers meta (meta muxing) on its source pad.
6. The qtioverlay plugin receives the muxed stream and overlays the bounding boxes on the VideoFrame using CL and produces gst buffers with overlays in its source pad.
7. waylandsink submits the video stream its receiving on its sinkpad to weston and finally weston renders the video stream on a local display device.

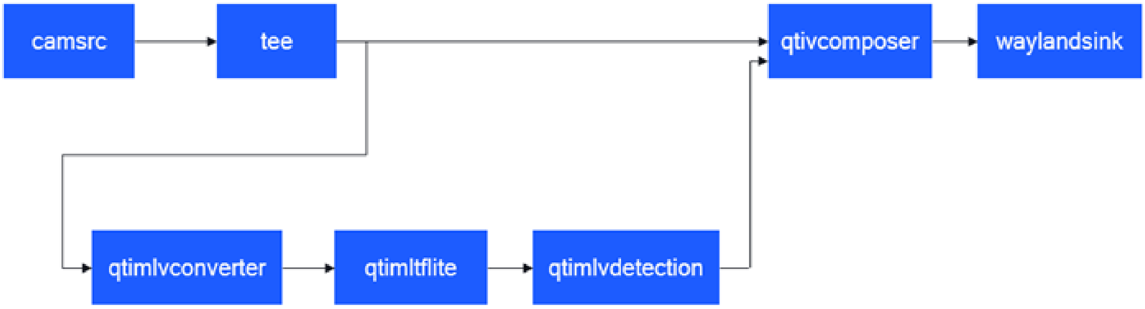
See the video stream captured by the camera source plugin and bounding boxes generated for the objects in that scene on local display device.

Command:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
waylandsink sync=false fullscreen=true \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection
threshold=75.0 results=10 module=yolov5m labels=/data/yolov5m.labels ! text/x-
raw ! queue ! metamux
```

To stop the use case, press **CTRL + C**.

Variant 2: Use qtivcomposer to mix original frame with bounding box mask



The use case is about using an yolov5m tflite model to do the following:

1. Identify objects in scene from video stream coming through camera source
2. Compose boundig boxes over objects detected and original video stream together using qtivcomposer
3. Display the results to a local display

The following is sequence of steps used to execute the use case:

1. The video stream is collected from camera source plugin and two copies are created
   - one is sent to qtivcomposer plugin to retain the video stream
   - Another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that yolov5m model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
   a. Loads the yolov5m model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the object detection results on its source pad
4. The postprocessing plugin for working on inference results from any object detection model ,qtimlvdetection, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvdetection plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of detection models.

   Here, in this use case, qtimlvdetection loads yolov5m submodule and produces video frames with only bounding boxes that can be overlaid on objects and sends them to sinkpad of qtivcomposer.

5. The qtivcomposer plugin receives the original video stream and video stream with bounding boxes on its sinkpads and produces on its sourcepad gst buffers with contents composed of video streams on its sinkpads.
6. waylandsink submits the video stream it is receiving on its sinkpad to weston and weston renders the video stream on a local display device.

See video stream captured by camera source plugin and bounding boxes drawn over allowed number of objects identified in that scene on local display device.

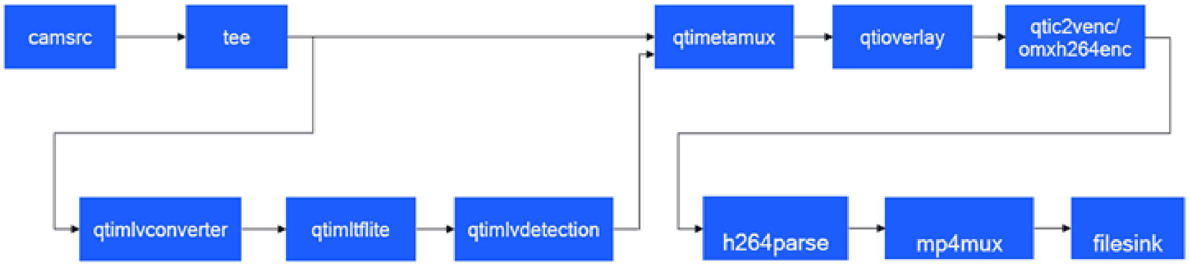
Command:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>" !
queue ! waylandsink sync=false fullscreen=true \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection
threshold=75.0 results=10 module=yolov5m labels=/data/yolov5m.labels !
video/x-raw,format=BGRA,width=640,height=360 ! queue ! mixer.
```

To stop the use case, press **CTRL + C**.

6.1.4 Single camera stream with object detection and encode

Variant 1: Use qtioverlay plugin to apply bounding box overlay



The use case is about using an yolov5m tflite model to do the following:

1. Identify object in scene from video stream coming through camera source
2. Overlay bounding boxes over the detected objects using overlaylib and encode this stream as h264 bitstream
3. Mux in a MP4 container and store it as an mp4 file

The following sequence describes the use case execution:

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to the qtimetamux plugin to retain the video stream
   - Another is sent to the ML inferencing pipeline

2. The preprocessing plugin, qtimlvconverter, does the following:
    a. Receives the video stream on its sink pad
    b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
    c. Converts the video stream to a tensor stream on its source pad that the yolov5m model can use for inferencing in a later part of pipeline
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
    a. Loads the yolov5m model
    b. Modifies the graph for the chosen delegate
    c. Receives the tensor stream on its sinkpad
    d. Executes the inference and produces tensor stream with the object detection results on its source pad
4. The postprocessing plugin for working on inference results from any object detection model ,qtimlvdetection, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

    The qtimlvdetection plugin applies the threshold to chose the number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of detection models.

    Here, in this use case, qtimlvdetection loads yolov5m submodule and produces results as structures of text and sends them to sinkpad of metamux.
5. The metamux plugin receives video stream and text stream with boundig box results corresponding to video stream on its sinkpads and produces gst buffers with contents of video stream from its sink pad, adding bounding boxes as GstVideoRegionOfInterest from data sinkpad to gst buffer's meta (meta muxing) on its source pad.
6. The qtioverlay plugin receives the muxed stream and overlays the bounding boxes on the VideoFrame using CL and produces gst buffers with overlays in its source pad.
7. The qtic2venc pluin applies parameters to each frame of the video stream it is receiving on its sinkpad and encodes it into bitstream and send it over its sourcepad.
8. h264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates containers format specification buffers
9. Filesink stores the resulting stream in the`/data/video.mp4` file.
10. Pull video.mp4 from device and play it in a media player application.

Command:

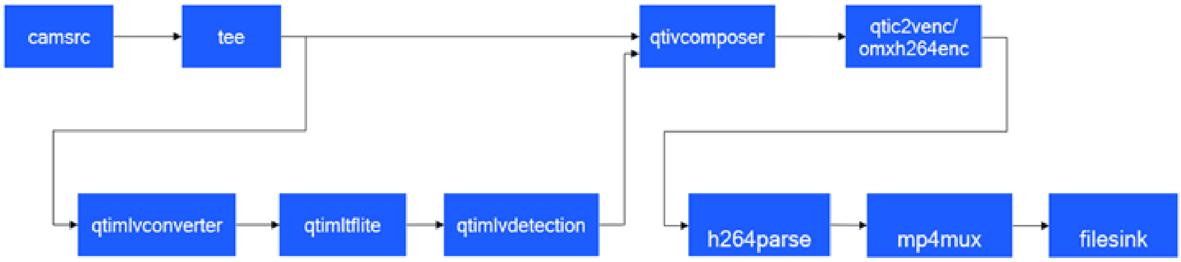
```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
qtic2venc target-bitrate=6000000 ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection
```

```
threshold=75.0 results=10 module=yolov5m labels=/data/yolov5m.labels ! text/x-
raw ! queue ! metamux
```

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
omxh264enc control-rate=max-bitrate target-bitrate=6000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection
threshold=75.0 results=10 module=yolov5m labels=/data/yolov5m.labels ! text/x-
raw ! queue ! metamux
```

To stop the use case, press **CTRL + C**.

Variant 2: Use qtivcomposer to mix original frame with bounding box mask



The use case is about using an yolov5m tflite model to do the following:

1. Identify objects in scene from video stream coming through camera source
2. Compose boundig boxes over objects detected and original video stream together using qtivcomposer
3. Encode this stream as h264 bitstream
4. Mux in an MP4 container and store it as an MP4 file

The following sequence describes the execution of the use case:

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to the qtivcomposer plugin to retain the video stream
   - Another is sent to the ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that the yolov5m model can use for inferencing in later part of pipeline

3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
    a. Loads the yolov5m model
    b. Modifies the graph for the chosen delegate
    c. Receives the tensor stream on its sinkpad
    d. Executes the inference and produces tensor stream with the object detection results on its source pad
4. The postprocessing plugin for working on inference results from any object detection model, qtimlvdetection, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

    The qtimlvdetection plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of detection models.

    Here, in this use case, qtimlvdetection loads the yolov5m submodule and produces video frames with only bounding boxes that can be overlaid on objects and sends them to sinkpad of qtivcomposer.
5. The qtivcomposer plugin receives original video stream and video stream with bounding boxes on it's sinkpads and produces on it's sourcepad gst buffers with contents composed of Video streams on it's sinkpads.
6. The qtic2venc plugin applies parameters to each frame of the video stream it is receiving on its sinkpad and encodes it into bitstream and sends it over its sourcepad.
7. h264parse adds additional information corresponding to the bitstream to the gstreamer buffer meta and the mp4mux plugin receives these buffers and creates containers format specification buffers
8. Filesink stores the resulting stream in a`/data/video.mp4` file.
9. Pull video.mp4 from device and play it in a mediaplayer application.

Command:

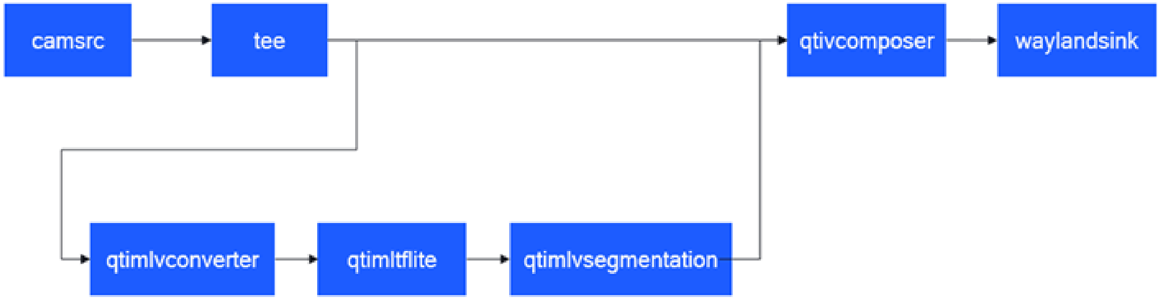
```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>" !
queue ! qtic2venc target-bitrate=6000000 ! h264parse ! queue ! mp4mux !
queue ! filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection
threshold=75.0 results=10 module=yolov5m labels=/data/yolov5m.labels !
video/x-raw,format=BGRA,width=640,height=360 ! queue ! mixer
```

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>" !
queue ! omxh264enc control-rate=max-bitrate target-bitrate=6000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse ! queue ! mp4mux ! queue !
```

```
filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/yolov5m-320x320-int8.tflite ! queue ! qtimlvdetection
threshold=75.0 results=10 module=yolov5m labels=/data/yolov5m.labels !
video/x-raw,format=BGRA,width=640,height=360 ! queue ! mixer
```

To stop the use case, press **CTRL + C**.

6.1.5 Single camera stream with image segmentation and display



The use case is about using the dv3_argmax_int32 tflite model to do the following:

1. Identify semantic segmentations in scene from video stream coming through camera source
2. Compose the semantics and original video stream together using qtivcomposer
3. Display the results on a local display

The following sequence describes the use case execution:

1. The video stream is collected from camera source plug-in and two copies are created:
   - one is sent to qtivcomposer plug-in to retain the video stream
   - Another is sent to ML inferencing pipeline
2. 3. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that dv3_argmax_int32 model can use for inferencing in a later part of pipeline
3. The ML inferencing plug-in for TensorflowLite runtime, qtimltflite, does the following:
   a. Loads the dv3_argmax_int32 model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the segmentation results on its source pad

4. The postprocessing plugin for working on inference results from any segmentation model ,qtimlvsegmentation converts the inference tensors it receives on its sinkpad into video formats that the multimedia plugins can understand later.

   The qtimlvsegmentation plugin produces the semantic segmentations for the frame the user is looking for. This plugin is capable of loading corresponding modules for a variety of segmentation models.

   Here, in this use case, qtimlvsegmentation loads deeplab-argmax submodule and produces video frames with segmentation masks and sends them to sinkpad of qtivcomposer.

5. The qtivcomposer plugin receives original video stream and video stream with segmentation mask on its sinkpads and produces on its sourcepad gst buffers with contents composed of video streams from the sinkpads.

6. waylandsink submits the video stream its receiving on its sinkpad to weston and weston renders the video stream on a local display device.

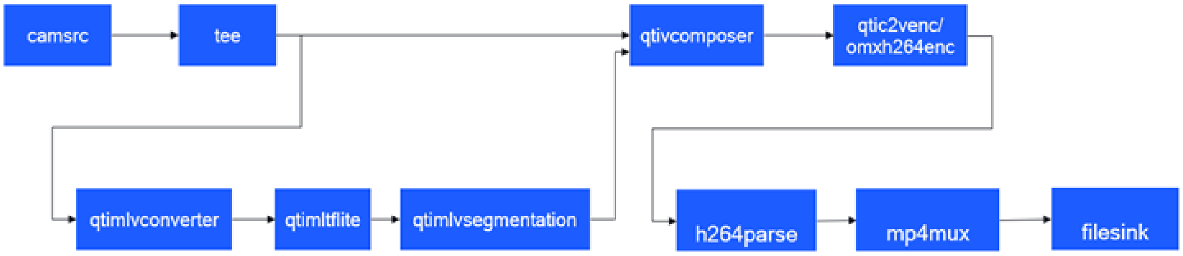
See the video stream captured by camera source plugin and segmentation masks drawn over objects/ components in that scene on the local display device.

Command:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>"
sink_1::alpha=0.5 ! queue ! waylandsink sync=false fullscreen=true \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=nnapi-dsp
model=/data/dv3_argmax_int32.tflite ! queue ! qtimlvsegmentation
module=deeplab-argmax labels=/data/dv3-argmax.labels ! video/x-
raw,width=256,height=144 ! queue ! mixer
```

To stop the use case, press **CTRL + C**.

6.1.6 Single camera stream with image segmentation and encode



The use case is about using the dv3_argmax_int32 TFLite model to do the following:

1. Identify the semantic segmentations in scene from video stream coming through camera source
2. Compose the semantics and original video stream together using qtivcomposer and encoding this stream as h264 bitstream, later muxing in an MP4 container
3. Storing the stream as an MP4 file

The following sequence describes the use case execution:

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to qtivcomposer plugin to retain the video stream
   - Another is sent to the ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that the dv3_argmax_int32 model can use to inference in a later part of the pipeline.
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
   a. Loads the dv3_argmax_int32 model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the segmentation results on its source pad
4. The postprocessing plugin for working on inference results from any segmentation model ,qtimlvsegmentation, converts the inference tensors it receives on it's sinkpad into video formats that our multimedia plugins can understand later.

   The qtimlvsegmentation plugin produces the semantic segmentations for the frame the user is looking for. This plugin is capable of loading corresponding modules for a variety of segmentation models.

   Here, in this use case, qtimlvsegmentation loads the deeplab-argmax submodule and produces video frames with segmentation masks and sends them to sinkpad of qtivcomposer.
5. The qtivcomposer plugin receives original video stream and video stream with segmentation mask on its sinkpads and produces on its sourcepad the gst buffers with contents composed of video streams from the sinkpads.
6. The qtic2venc plugin applies parameters to each frame of the video stream it is receiving on its sinkpad and encodes it into bitstream, and sends it over its sourcepad.
7. h264parse adds the additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers, and creates containers to format the specification buffers
8. Filesink stores the resulting stream in a`/data/video.mp4` file.
9. Pull video.mp4 from device and play it on a media player application.

Command:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>"
sink_1::alpha=0.5 ! queue ! qtic2venc target-bitrate=6000000 ! h264parse !
```

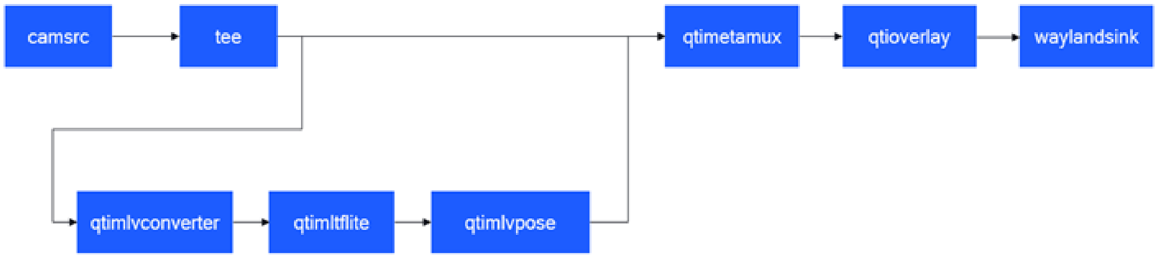
```
queue ! mp4mux ! queue ! filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=nnapi-dsp
model=/data/dv3_argmax_int32.tflite ! queue ! qtimlvsegmentation
module=deeplab-argmax labels=/data/dv3-argmax.labels ! video/x-
raw,width=256,height=144 ! queue ! mixer.
```

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>"
sink_1::alpha=0.5 ! queue ! omxh264enc control-rate=max-bitrate target-
bitrate=6000000 interval-intraframes=29 periodicity-idr=1 ! h264parse !
queue ! mp4mux ! queue ! filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=nnapi-dsp
model=/data/dv3_argmax_int32.tflite ! queue ! qtimlvsegmentation
module=deeplab-argmax labels=/data/dv3-argmax.labels ! video/x-
raw,width=256,height=144 ! queue ! mixer
```

To stop the use case, press **CTRL + C**.

6.1.7 Single camera stream with pose estimation and display

Variant 1: Use qtioverlay plugin to apply pose estimation overlay



The use case is about using a MobileNet TFLite model to do the following:

- Identify poses of people in scenes from video stream coming through camera source
- Overlay available poses using overlaylib and show this stream on local display device

1. The video stream is collected from camera source plugin and two copies are created:
    - One is sent to qtimetamux plugin to retain the video stream
    - Another is sent to ML inferencing pipeline

2. The preprocessing plugin, qtimlvconverter, does the following:
    a. Receives the video stream on its sink pad
    b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
    c. Converts the video stream to a tensor stream on its source pad that the PoseNet model can use for inferencing in later part of pipeline.
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
    a. Loads the PoseNet model
    b. Modifies the graph for the chosen delegate
    c. Receives the tensor stream on its sinkpad
    d. Executes the inference and produces tensor stream with the pose estimation results on its source pad
4. The postprocessing plugin for working on inference results from PoseNet model, qtimlvpose, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

    The qtimlvpose plugin applies the threshold to the chosen number of results the user is looking for. This plugin is loads the corresponding modules for a variety of pose estimation models.

    Here, in this use case, qtimlvpose loads the PoseNet submodule and produces results as structures of text and sends them to the sinkpad of metamux.
5. The metamux plugin receives the video stream and text stream with pose results corresponding to video stream on its sinkpads and produces gst buffers with contents of video stream from the sink pad, adding poses from data sinkpad to gst buffer's meta (meta muxing) on its source pad.
6. The qtioverlay plugin receives the muxed stream and overlays the poses on the VideoFrame using CL and produces gst buffers with overlays in its source pad.
7. waylandsink submits the video stream it is receiving on its sinkpad to weston and weston renders the video stream on a local display device.

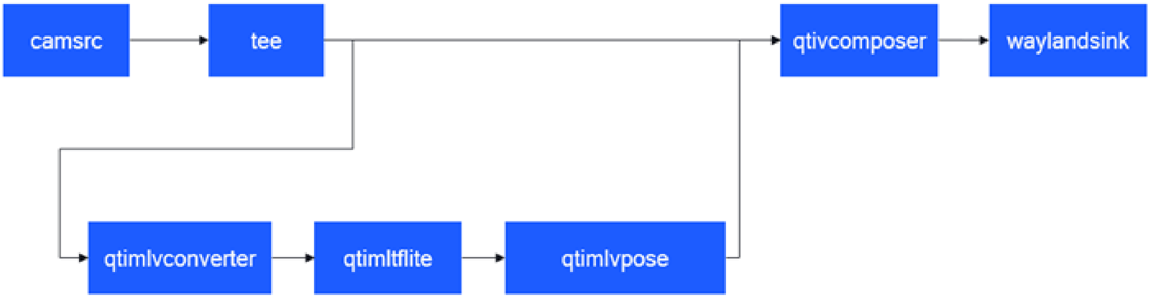
See video stream captured by THE camera source plugin and poses generated for multiple people in that scene on a local display device.

Command:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
waylandsink sync=false fullscreen=true \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/posenet_mobilenet_v1_075_481_641_quant.tflite ! queue !
qtimlvpose threshold=51.0 results=2 module=posenet labels=/data/
posenet.labels ! text/x-raw ! queue ! metamux
```

To stop the use case, press **CTRL + C**.

Variant 2: Use qtivcomposer to mix original frame with pose estimation mask



The use case is about using a PoseNet TFLite model to do the following:

- Classify scenes from a video stream coming through camera source
- Compose the poses and video stream together using qtivcomposer and show it on a local display device

1. The video stream is collected from the camera source plugin and two copies are created:
   - one is sent to the qtivcomposer plugin to retain the video stream
   - Another is sent to the ML inferencing branch in the pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that PoseNet model can use for inferencing in a later part of pipeline.
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
   a. Loads the PoseNet model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the inference results on its source pad
4. The postprocessing plugin for working on inference results from a pose estimation model ,qtimlvpose, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvpose plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of pose estimation models.

   Here, in this use case, qtimlvpose loads PoseNet submodule and produces results as video frames with poses drawn and sends them to sinkpad of the qtivcomposer.

5. The qtivcomposer plugin receives the original video stream and video stream of poses on its sinkpads and produces on its sourcepad the gst buffers with contents composed of video streams from its sinkpads
6. Waylandsink submits the video stream it is receiving on it's sinkpad to weston and weston renders the video stream on a local display device.

See the video stream captured by camera source plugin and poses generated for multiple people in that scene on a local display device.

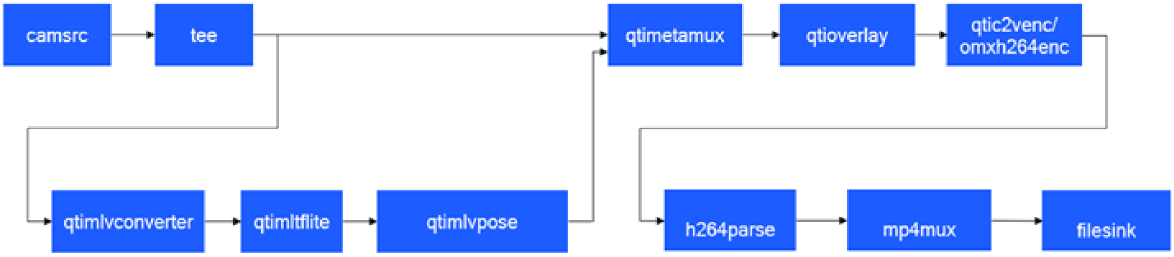
Command:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>" !
queue ! waylandsink sync=false fullscreen=true \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/posenet_mobilenet_v1_075_481_641_quant.tflite ! queue !
qtimlvpose threshold=51.0 results=2 module=posenet labels=/data/
posenet.labels ! video/x-raw,format=BGRA,width=640,height=360 ! queue ! mixer
```

To stop the use case, press **CTRL + C**.

6.1.8 Single camera stream with pose estimation and encode

Variant 1: Use qtioverlay plugin to apply pose estimation overlay



The use case is about using a MobileNet TFLite model to do the following:

1. Identify poses of people in scenes from video stream coming through camera source
2. Overlay the available poses using overlaylib and encode this stream as a h264 bitstream
3. Mux in a MP4 container and store it as a MP4 file

The following sequence describes the use case execution:

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to the qtimetamux plugin to retain the video stream
   - Another is sent to the ML inferencing pipeline

2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that the PoseNet model can use for inferencing in a later part of pipeline
3. 4. Our ML inferencing plugin for TensorflowLite runtime, qtimltflite, loads the posenet model, modifies the graph for the chosen delegate, receives the tensor stream on it's sinkpad, executes the inference and produces tensor stream with the pose estimation results on it's source pad.
4. The postprocessing plugin for working on inference results from PoseNet model ,qtimlvpose, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvpose plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of pose estimation models.

   Here,in this use case, qtimlvpose loads PoseNet submodule and produces results as structures of text and sends them to sinkpad of metamux.
5. The metamux plugin receives the video stream and text stream with pose results corresponding to video stream on its sinkpads and produces gst buffers with contents of video stream on its sink pad, adding poses from data sinkpad to gst buffer's meta (meta muxing) on its source pad.
6. The qtioverlay plugin receives the muxed stream and overlays the poses on the VideoFrame using CL and produces gst buffers with overlays in its source pad.
7. The qtic2venc plugin applies parameters to each frame of the video stream it is receiving on its sinkpad and encodes it into bitstream and sends it over its sourcepad.
8. h264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates containers format specification buffers
9. Filesink stores the resulting stream in a`/data/video.mp4` file.
10. Pull video.mp4 from device and play it on a media player application.

Command:

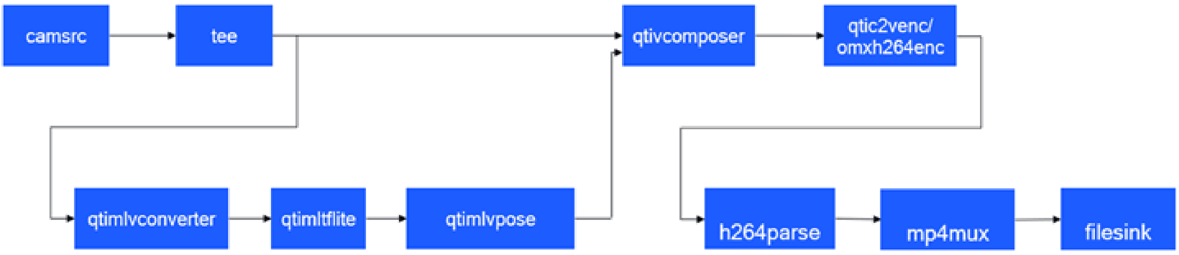
```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
qtic2venc  target-bitrate=6000000 ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/posenet_mobilenet_v1_075_481_641_quant.tflite ! queue !
qtimlvpose threshold=51.0 results=2 module=posenet labels=/data/
posenet.labels ! text/x-raw ! queue ! metamux.
```

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
```

```
omxh264enc control-rate=max-bitrate target-bitrate=6000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/posenet_mobilenet_v1_075_481_641_quant.tflite ! queue !
qtimlvpose threshold=51.0 results=2 module=posenet labels=/data/
posenet.labels ! text/x-raw ! queue ! metamux
```

To stop the use case, press **CTRL + C**.

Variant 2: Use qtivcomposer to mix original frame with pose estimation mask



The use case is about using a PoseNet TFLite model to do the following:

- Classify scenes from the video stream coming through camera source
- Compose the poses and video stream together using qtivcomposer
- Encode this stream as a h264 bitstream
- Mux in a MP4 container and storing it as a MP4 file

1. The video stream is collected from the camera source plugin and two copies are created:
   - one is sent to qtivcomposer plugin to retain the video stream
   - Another is sent to ML inferencing branch in the pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that PoseNet model can use for inferencing in a later part of pipeline
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
   a. Loads the Posenet model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the inference results on its source pad

4. The postprocessing plugin for working on inference results from a pose estimation model ,qtimlvpose, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvpose plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of pose estimation models.

   Here, in this use case, qtimlvpose loads PoseNet submodule and produces results as video frames with poses drawn, and sends them to sinkpad of qtivcomposer.
5. The qtivcomposer plugin receives the original video stream and sideo stream of poses on its sinkpads and produces on its sourcepad the gst buffers with contents composed of video streams from its sinkpads
6. The qtic2venc plugin applies parameters to each frame of the video stream it's receiving on its sinkpad and encodes it into bitstream, and sends it over its sourcepad.
7. h264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates container's format specification buffers
8. Filesink stores the resulting stream in a`/data/video.mp4` file.
9. Pull video.mp4 from device and play it on a media player application.

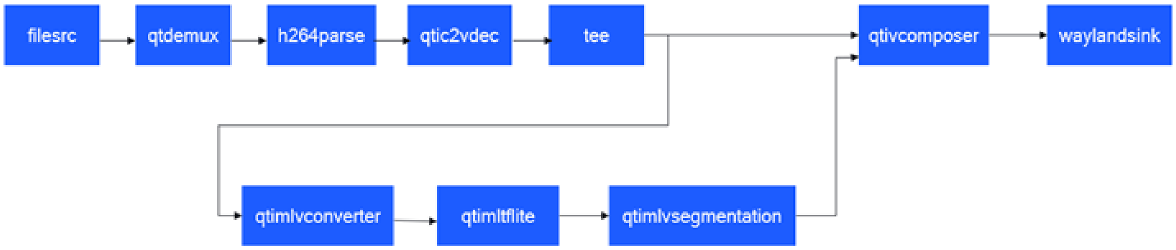
Commands:

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>" !
queue ! qtic2venc target-bitrate=6000000 ! h264parse ! queue ! mp4mux !
queue ! filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/posenet_mobilenet_v1_075_481_641_quant.tflite ! queue !
qtimlvpose threshold=51.0 results=2 module=posenet labels=/data/
posenet.labels ! video/x-raw,format=BGRA,width=640,height=360 ! queue ! mixer
```

```
gst-launch-1.0 -e --gst-debug=2 \
qtiqmmfsrc name=camsrc ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee name=split
\
split. ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>" !
queue ! omxh264enc control-rate=max-bitrate target-bitrate=6000000 interval-
intraframes=29 periodicity-idr=1 ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 \
split. ! queue ! qtimlvconverter ! queue ! qtimltflite delegate=hexagon
model=/data/posenet_mobilenet_v1_075_481_641_quant.tflite ! queue !
qtimlvpose threshold=51.0 results=2 module=posenet labels=/data/
posenet.labels ! video/x-raw,format=BGRA,width=640,height=360 ! queue ! mixer
```

To stop the use case, press **CTRL + C**.

6.1.9 Single video file stream with mono depth estimation



This use case is about using video stream from a single camera or a decoded stream from a filesource and using the TFLite model midas_v2 to compute depth map, and show it on local display

1. 1. The pipeline reads sample file using filesrc and uses qtic2vdec decoder plugin to produce decode video stream.
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that midas_v2 model can use for inferencing in a later part of pipeline
3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
   a. Loads the midas_v2 model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the inference results on its source pad
4. The postprocessing plugin for working on inference results from any segmentation model, qtimlvsegmentation, converts the inference tensors it receives on its sinkpad into video formats that the multimedia plugins can understand later.

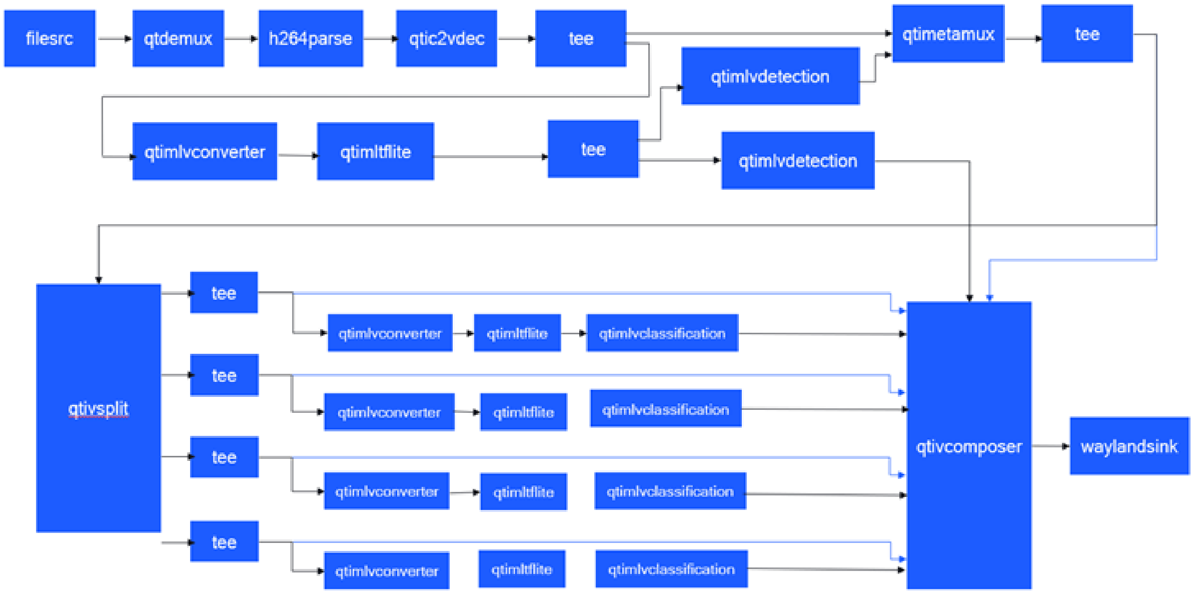
   The qtimlvsegmentation plugin produces the depth map for the frame the user is looking for. This plugin is capable of loading corresponding modules for a variety of segmentation models.

   Here, in this use case, qtimlvsegmentation loads midas-v2 submodule and produces video frames with depth masks and sends them to sinkpad of qtivcomposer.
5. qtivcomposer composes frames with contents from its sinkpads and pushes gst buffers with them on its source pad
6. Waylandsink submits the video stream it is receiving on its sinkpad to weston and weston renders the video stream on a local display device.
7. See video stream captured by camera source plugin and corresponding depth map generated for that scene on local display device.

Command:

```
ulimit -n 4096 && gst-launch-1.0 -e --gst-debug=2,fpsdisplaysink:6 \
qtivcomposer name=mixer sink_1::dimensions="<384, 216>" \
mixer. ! queue ! fpsdisplaysink sync=false signal-fps-measurements=true text-
overlay=false video-sink="waylandsink fullscreen=true sync=false sync=false" \
filesrc location=/data/Driving_720p_180s_30FPS.MOV ! qtdemux ! queue !
h264parse ! qtic2vdec ! queue ! tee name=t_split ! queue ! mixer. \
t_split. ! queue ! qtimlvconverter mean="<123.675,116.28,103.53>"
sigma="<58.395,57.12,57.375>" ! queue ! qtimltflite delegate=gpu model=/data/
midas_v2.tflite ! queue ! qtimlvsegmentation module=midas-v2 labels=/data/
monodepth.labels ! video/x-raw,width=256,height=144 ! queue ! mixer
```

6.1.10 Daisy chain object detection and image classification on single video file stream



The use case uses multiple QIM SDK plugins to showcase the complex scenario where the objects being seen in a scene are classified and shown to the user.

1. The video stream is collected from camera source plugin and two copies are created:
   - one is sent to the qtimetamux plugin to retain the video stream
   - Another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that the yolov5m model can use for inferencing in later part of pipeline

3. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
   a. Loads the yolov5m model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the object detection results on its source pad

   A copy of the tensor stream is sent to one qtimlvdetection plugin and one to another.

4. The postprocessing plugin for working on inference results from any object detection model ,qtimlvdetection, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvdetection plugin applies the threshold to chose the number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of detection models.

   Here,in this use case, qtimlvdetection loads yolov5m submodule and produces results as structures of text and sends them to sinkpad of metamux.

5. The metamux plugin receives video stream and text stream with bounding box results corresponding to video stream on its sinkpads and produces gst buffers with contents of video stream from its sink pad, adding bounding boxes as GstVideoRegionOfInterest from data sinkpad to gst buffer's meta (meta muxing) on its source pad.

6. Second qtimlvconverter creates bounding boxes as video streams and pushes to qtivcomposer for it to overlay on original video frames.

7. Two copies of streams from metamux are created, one shared to qtivcomposer as original frame and second stream sent to qtivsplit.

8. The qtivsplit plugin creates crops of objects in the frame and push each crop on one of its source pads and two copies of each crop are created, one pushed to qtivcomposer for composition and another to qtimlvconverter for classification branch.

9. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that MobileNet model can use for inferencing in later part of pipeline.

10. The ML inferencing plugin for TensorflowLite runtime, qtimltflite, does the following:
    a. Loads the MobileNet model
    b. Modifies the graph for the chosen delegate
    c. Receives the tensor stream on its sinkpad
    d. Executes the inference and produces tensor stream with the inference results on its source pad

11. The postprocessing plugin for working on inference results from a classification model ,qtimlvclassification, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

    The qtimlvclassification plugin applies the threshold to chose the number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of classification models.

    Here, in this use case, qtimlvclassification loads MobileNet submodule and produces results as video frames with classification labels and sends them to sinkpad of qtivcomposer.

12. qtivcomposer composes original frame with bounding boxes overlaid on objects, each object with its classification labels.
13. Waylandsink receives the buffers and submits to weston.
14. Weston renders on a local display device.

Command:

```
ulimit -n 4096 && gst-launch-1.0 -e --gst-debug=2,fpsdisplaysink:6 \
qtimltflite name=tflite_yolov5 delegate=hexagon model=/data/yolov5m-320x320-
int8.tflite \
qtimltflite name=tflite_Mobilenet_1 delegate=hexagon model=/data/
mobilenet_v2_1.0_224_quant.tflite \
qtimltflite name=tflite_Mobilenet_2 delegate=hexagon model=/data/
mobilenet_v2_1.0_224_quant.tflite \
qtimltflite name=tflite_Mobilenet_3 delegate=hexagon model=/data/
mobilenet_v2_1.0_224_quant.tflite \
qtimltflite name=tflite_Mobilenet_4 delegate=hexagon model=/data/
mobilenet_v2_1.0_224_quant.tflite \
qtimlvconverter name=ml_convert_0 ! queue max-size-bytes=0 max-size-time=0 !
tflite_yolov5. tflite_yolov5. ! queue max-size-bytes=0 max-size-time=0 ! tee
name=t_split_1 \
qtivcomposer name=mixer \
sink_0::position="<0, 0>" sink_0::dimensions="<1280, 720>" \
sink_1::position="<0, 0>" sink_1::dimensions="<1280, 720>" \
sink_2::position="<0, 0>" sink_2::dimensions="<384, 216>" \
sink_3::position="<896, 0>" sink_3::dimensions="<384, 216>" \
sink_4::position="<0, 504>" sink_4::dimensions="<384, 216>" \
sink_5::position="<896, 504>" sink_5::dimensions="<384, 216>" \
sink_6::position="<0, 0>" sink_6::dimensions="<384, 40>" \
sink_7::position="<896, 0>" sink_7::dimensions="<384, 40>" \
sink_8::position="<0, 504>" sink_8::dimensions="<384, 40>" \
sink_9::position="<896, 504>" sink_9::dimensions="<384, 40>" \
mixer. ! video/x-raw\(memory:GBM\),format=NV12,compression=ubwc ! queue max-
size-bytes=0 max-size-time=0 ! fpsdisplaysink sync=false signal-fps-
measurements=true text-overlay=false video-sink="waylandsink sync=false
fullscreen=true" \
filesrc location=/data/Street_Side_720p_180s_30FPS.MOV ! qtdemux !
h264parse ! qtic2vdec ! video/x-raw\(memory:GBM
\),format=NV12,compression=ubwc ! queue max-size-bytes=0 max-size-time=0 !
tee name=v_split_1 ! queue max-size-bytes=0 max-size-time=0 ! metamux1.
```

```
v_split_1. ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_0. \
t_split_1. ! queue max-size-bytes=0 max-size-time=0 ! qtimlvdetection
threshold=75.0 results=4 module=yolov5m labels=/data/yolov5m.labels ! text/x-
raw ! queue max-size-bytes=0 max-size-time=0 ! qtimetamux name=metamux1 !
queue max-size-bytes=0 max-size-time=0 ! tee name=t_split_2 ! queue max-size-
bytes=0 max-size-time=0 ! mixer. \
t_split_1. ! queue max-size-bytes=0 max-size-time=0 ! qtimlvdetection
threshold=75.0 results=5 module=yolov5m labels=/data/yolov5m.labels ! video/x-
raw,width=512,height=288 ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
t_split_2. ! queue max-size-bytes=0 max-size-time=0 ! qtivsplit name=vsplit1
mode=roi \
vsplit1. ! video/x-raw,format=BGRA ! queue max-size-bytes=0 max-size-time=0 !
tee name=split_1 ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_1.
split_1. ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
vsplit1. ! video/x-raw,format=BGRA ! queue max-size-bytes=0 max-size-time=0 !
tee name=split_2 ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_2.
split_2. ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
vsplit1. ! video/x-raw,format=BGRA ! queue max-size-bytes=0 max-size-time=0 !
tee name=split_3 ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_3.
split_3. ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
vsplit1. ! video/x-raw,format=BGRA ! queue max-size-bytes=0 max-size-time=0 !
tee name=split_4 ! queue max-size-bytes=0 max-size-time=0 ! ml_convert_4.
split_4. ! queue max-size-bytes=0 max-size-time=0 ! mixer. \
qtimlvconverter name=ml_convert_1 ! queue max-size-bytes=0 max-size-time=0 !
tflite_Mobilenet_1. tflite_Mobilenet_1. ! queue max-size-bytes=0 max-size-
time=0 ! mlclass_1. \
qtimlvconverter name=ml_convert_2 ! queue max-size-bytes=0 max-size-time=0 !
tflite_Mobilenet_2. tflite_Mobilenet_2. ! queue max-size-bytes=0 max-size-
time=0 ! mlclass_2. \
qtimlvconverter name=ml_convert_3 ! queue max-size-bytes=0 max-size-time=0 !
tflite_Mobilenet_3. tflite_Mobilenet_3. ! queue max-size-bytes=0 max-size-
time=0 ! mlclass_3. \
qtimlvconverter name=ml_convert_4 ! queue max-size-bytes=0 max-size-time=0 !
tflite_Mobilenet_4. tflite_Mobilenet_4. ! queue max-size-bytes=0 max-size-
time=0 ! mlclass_4. \
qtimlvclassification name=mlclass_1 threshold=51.0 results=2 module=mobilenet
labels=/data/mobilenet.labels ! video/x-raw,width=384,height=40 ! queue max-
size-bytes=0 max-size-time=0 ! mixer. \
qtimlvclassification name=mlclass_2 threshold=51.0 results=2 module=mobilenet
labels=/data/mobilenet.labels ! video/x-raw,width=384,height=40 ! queue max-
size-bytes=0 max-size-time=0 ! mixer. \
qtimlvclassification name=mlclass_3 threshold=51.0 results=2 module=mobilenet
labels=/data/mobilenet.labels ! video/x-raw,width=384,height=40 ! queue max-
size-bytes=0 max-size-time=0 ! mixer. \
qtimlvclassification name=mlclass_4 threshold=51.0 results=2 module=mobilenet
labels=/data/mobilenet.labels ! video/x-raw,width=384,height=40 ! queue max-
size-bytes=0 max-size-time=0 ! mixer
```

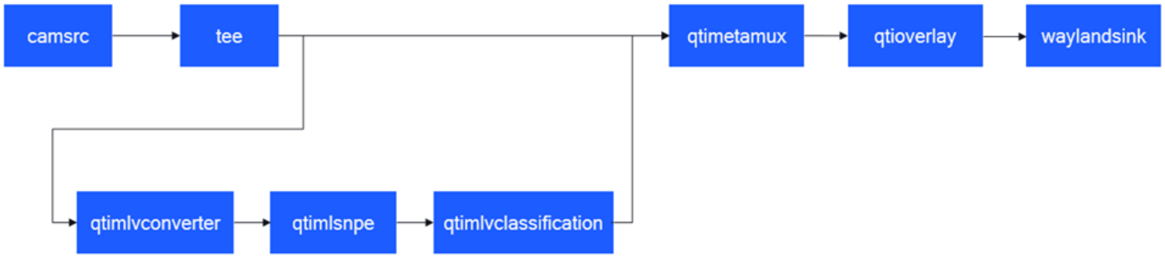
6.2 Qualcomm Neural Processing SDK use cases

Qualcomm Neural Processing SDK was formerly known as Qualcomm Snapdragon Neural Processing Engine (SNPE).

NOTE The deep learning container (DLC) models used in the pipelines are available with the SNPE SDK release.

6.2.1 Single camera stream with image classification and display with MobileNet v1

Use qtioverlay plug-in to apply classification overlay



The use case is about using a MobileNet quantaware model to do the following:

- Classify scenes from video stream coming through camera source
- Overlay classification labels using overlaylib
- Display the results to a local display

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to qtimetamux plugin to retain the video stream
   - Another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that MobileNet model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the MobileNet model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the inference results on its source pad

4. The postprocessing plugin for working on inference results from a classification model, qtimlvclassification, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvclassification plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of classification models.

   Here, in this use case, qtimlvclassification loads MobileNet submodule and produces results as structures of text and sends them to sinkpad of metamux.

5. The metamux plugin receives the video stream and text stream with classification results corresponding to video stream on its sinkpads and produces gst buffers with contents of Video stream on its sink pad. It adds classification result from data sinkpad to gst buffer's meta (meta muxing) on its source pad.

6. The qtioverlay plugin receives the muxed stream and overlays the classification labels on the VideoFrame using CL and produces gst buffers with overlays in its source pad.

7. Waylandsink submits the video stream it is receiving on it's sinkpad to weston and finally weston renders the video stream on a local display device.

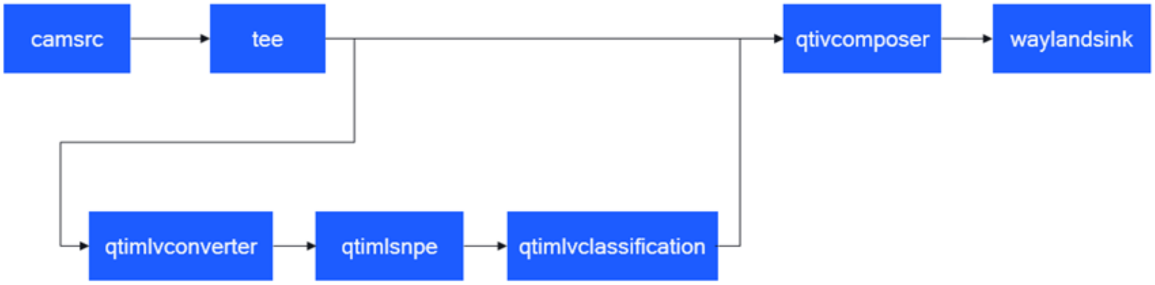
8. 9. One can see video stream captured by camera source plugin and possible classifications generated for that scene on local display device.

Commands:

```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
waylandsink sync=false fullscreen=true split. ! queue ! qtimlvconverter !
queue ! qtimlsnpe delegate=gpu model=/data/
tf2_10_axis_quant_public_cnns_cnns_mobilenet_v1_quantaware_batch_1_quant.dlc !
 queue ! qtimlvclassification threshold=60.0 results=3 module=mobilenet
labels=/data/mobilenet.labels ! text/x-raw ! queue ! metamux.
```

To stop the use case, press **CTRL + C**.

Use qtivcomposer to mix original frame with classification mask



The use case is about using a MobileNet quantaware model to do the following:

- Classify scenes from a video stream coming through camera source
- Compose classification labels and video stream together using qtivcomposer
- Display the results to a local display

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to qtivcomposer plugin to retain the video stream
   - Another is sent to ML inferencing branch in the pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on it's source pad that MobileNet model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the MobileNet model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the inference results on its source pad
4. The postprocessing plugin for working on inference results from a classification model, qtimlvclassification, is responsible for converting the inference tensors it receives on it's sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvclassification plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of classification models.

   Here, in this use case, qtimlvclassification loads MobileNet submodule and produces results as video frames with classification labels and sends them to sinkpad of qtivcomposer.
5. The qtivcomposer plugin receives original video stream and video stream with classification results on its sinkpads and produces on its sourcepad gst buffers with contents composed of video streams from its sinkpads.
6. Waylandsink submits the video stream it is receiving on its sinkpad to Weston and Weston renders the video stream on a local display device.

   See the video stream captured by camera source plugin and possible classifications generated for that scene on local display device.

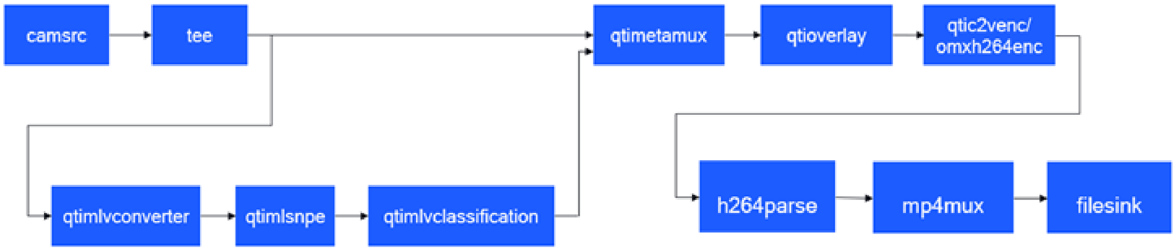
Commands:

```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtivcomposer name=mixer sink_1::position="<50, 50>"
sink_1::dimensions="<368, 64>" ! queue ! waylandsink sync=false
fullscreen=true split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe
delegate=gpu model=/data/
tf2_10_axis_quant_public_cnns_cnns_mobilenet_v1_quantaware_batch_1_quant.dlc !
 queue ! qtimlvclassification threshold=60.0 results=3 module=mobilenet
labels=/data/mobilenet.labels ! video/x-raw,format=BGRA,width=368,height=64 !
queue ! mixer.
```

To stop the use case, press **CTRL + C**.

6.2.2 Single camera stream with image classification and encode with MobileNet v1

Variant 1: Use qtioverlay plugin to apply classification overlay



The use case is about using a MobileNet model to do the following:

- Classify scenes from video stream coming through camera source
- Overlay classification labels using overlaylib
- Encode this stream as h264 bitstream and mux in an mp4 container
- Store it as an mp4 file

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to the qtimetamux plugin to retain the video stream
   - Another is sent to the ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that MobileNet model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the MobileNet model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the inference results on its source pad
4. The postprocessing plugin for working on inference results from a classification model, qtimlvclassification, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvclassification plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of classification models.

Here, in this use case, qtimlvclassification loads MobileNet submodule and produces results as structures of text and sends them to sinkpad of metamux.

5. The metamux plugin receives video stream and text stream with classification results corresponding to video stream on its sinkpads and produces gst buffers with contents of video stream on its sink pad, adding classification result from data sinkpad to gst buffer's meta (meta muxing) on its source pad.
6. The qtioverlay plugin receives the muxed stream and overlays the classification labels on the VideoFrame using CL and produces gst buffers with overlays in its source pad.
7. The qtic2venc plugin applies parameters to each frame of the video stream it is receiving on its sinkpad and encodes it into bitstream and send it over its sourcepad.
8. H264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates containers format specification buffers
9. Filesink stores the resulting stream in a the `/data/video.mp4` file.

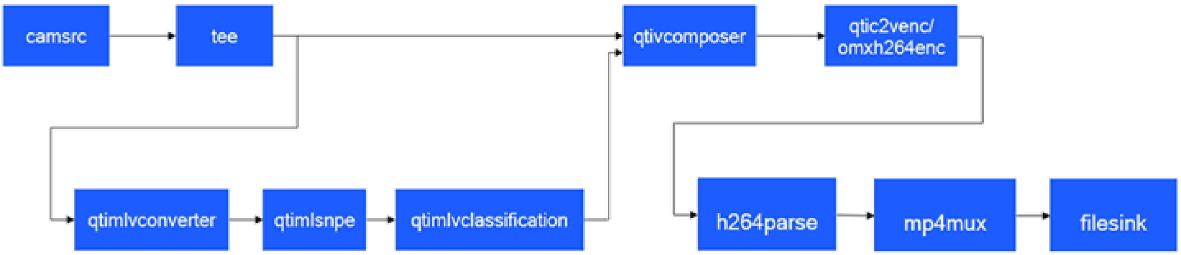
Pull video.mp4 from device and play it on a media player application.

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
qtic2venc target-bitrate=6000000 ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 split. ! queue ! qtimlvconverter ! queue !
qtimlsnpe delegate=gpu model=/data/
tf2_10_axis_quant_public_cnns_cnns_mobilenet_v1_quantaware_batch_1_quant.dlc !
 queue ! qtimlvclassification threshold=60.0 results=3 module=mobilenet
labels=/data/mobilenet.labels ! text/x-raw ! queue ! metamux
```

To stop the use case, press **CTRL + C**.

Variant 2: Use qtivcomposer to mix original frame with classification mask



The use case is about using a MobileNet quantaware model to do the following:

- Classify scenes from video stream coming through camera source
- Compose classification labels and video stream together using qtivcomposer
- Encode this stream as a h264 bitstream and mux in an mp4 container
- Store it as an mp4 file

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to the qtivcomposer plugin to retain the video stream
   - Another is sent to ML inferencing branch in the pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that MobileNet model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the MobileNet model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on it's sinkpad, executes the inference and produces tensor stream with the inference results on it's source pad.
4. The postprocessing plugin for working on inference results from a classification model, qtimlvclassification converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvclassification plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of classification models.

   Here, in this use case, qtimlvclassification loads MobileNet submodule and produces results as video frames with classification labels and sends them to sinkpad of qtivcomposer.
5. The qtivcomposer plugin receives the original video stream and video stream with classification results on its sinkpads and produces on its sourcepad gst buffers with contents composed of video streams on its sinkpads
6. The qtic2venc plugin applies parameters to each frame of the video stream it is receiving on its sinkpad and encodes it into bitstream and send it over its sourcepad.
7. H264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates containers format specification buffers
8. The filesink stores the resulting stream in the `/data/video.mp4` file.

The user can pull video.mp4 from device and play it on a media player application.

Command:

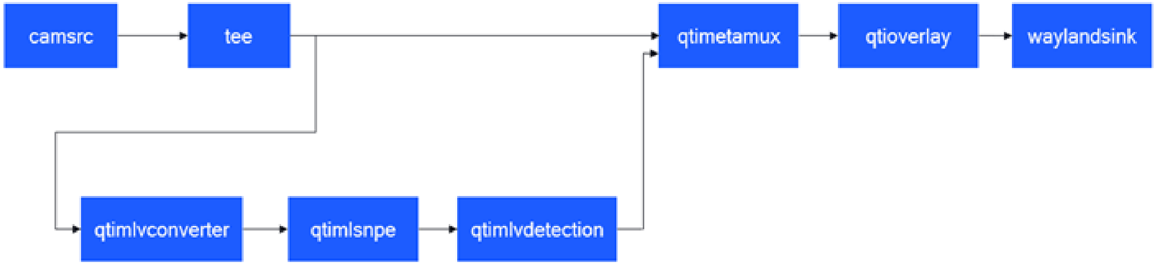
```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtivcomposer name=mixer sink_1::position="<50, 50>"
sink_1::dimensions="<368, 64>" ! queue ! qtic2venc target-bitrate=6000000 !
h264parse ! queue ! mp4mux ! queue ! filesink location=/data/video.mp4
split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=gpu model=/data/
tf2_10_axis_quant_public_cnns_cnns_mobilenet_v1_quantaware_batch_1_quant.dlc !
 queue ! qtimlvclassification threshold=60.0 results=3 module=mobilenet
```

```
labels=/data/mobilenet.labels ! video/x-raw,format=BGRA,width=368,height=64 !
queue ! mixer.
```

To stop the use case, press **CTRL + C**.

6.2.3 Single camera stream with object detection and display with MobileNet v2 SSD

Variant 1: Use qtioverlay plugin to apply bounding box overlay



The use case is about using a mobilenet_v2_ssd model to do the following:

- Identify an object in scene from video stream coming through camera source
- overlay bounding boxes over the detected objects using overlaylib
- Display the results on a local display

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to qtimetamux plugin to retain the video stream
   - Another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that mobilenet_v2_ssd model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the object detection results on its source pad

4. The postprocessing plugin for working on inference results from any object detection model, qtimlvdetection, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvdetection plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of detection models.

   Here,in this use case, qtimlvdetection loads ssd-mobilenet submodule and produces results as structures of text and sends them to sinkpad of metamux.

5. The metamux plugin does the following:

   a. Receives video stream and text stream of bounding box results corresponding to video stream on its sinkpads

   b. Produces gst buffers with contents of video stream from its sink pad, adding bounding boxes as GstVideoRegionOfInterest from data sinkpad to gst buffer's meta (meta muxing) on its source pad

6. The qtioverlay plugin receives the muxed stream and overlays the bounding boxes on the VideoFrame using CL and produces gst buffers with overlays in its source pad.

7. Waylandsink submits the video stream it's receiving on it's sinkpad to weston and weston renders the video stream on a local display device.

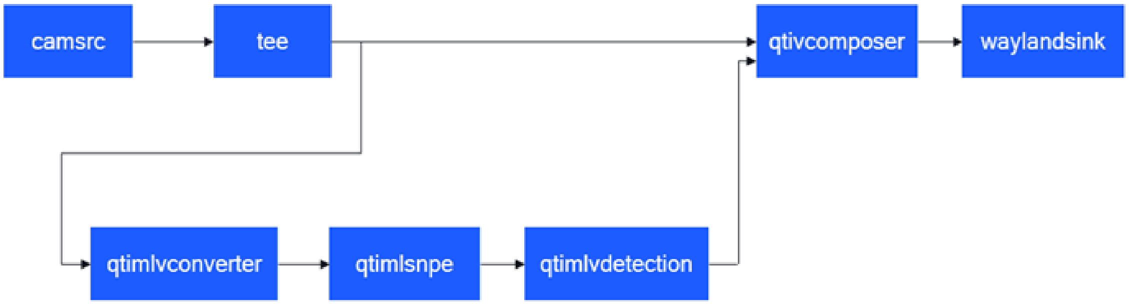
See video stream captured by camera source plugin and bounding boxes generated for the objects in that scene on local display device.

Commands:

```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
waylandsink sync=false fullscreen=true split. ! queue ! qtimlvconverter !
queue ! qtimlsnpe delegate=gpu model=/data/
tf11_public_cnns_cnns_mobilenet_v2_ssd_quant_aware_batch_1_quant.dlc
layers="<Postprocessor/BatchMultiClassNonMaxSuppression>" ! queue !
qtimlvdetection threshold=75.0 results=10 module=ssd-mobilenet labels=/data/
ssd-mobilenet.labels ! text/x-raw ! queue ! metamux.
```

To stop the use case, press **CTRL + C**.

Variant 2: Use qtivcomposer to mix original frame with bounding box mask



The use case is about using mobilenet_v2_ssd to do the following:

- Identify objects in scene from video stream coming through camera source
- Compose bounding boxes over objects detected and original video stream together using qtivcomposer
- Display the results on a local display

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to qtivcomposer plugin to retain the video stream
   - Another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that mobilenet_v2_ssd model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the mobilenet_v2_ssd model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the object detection results on its source pad
4. The postprocessing plugin for working on inference results from any object detection model ,qtimlvdetection, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvdetection plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of detection models.

   Here, in this use case, qtimlvdetection loads the ssd-mobilenet submodule and produces video frames with only bounding boxes that can be overlaid on objects and sends them to sinkpad of qtivcomposer.
5. The qtivcomposer plugin receives original video stream and video stream with bounding boxes on its sinkpads and produces on its sourcepad gst buffers with contents composed of video streams on its sinkpads.
6. Waylandsink submits the video stream its receiving on its sinkpad to weston and weston renders the video stream on a local display device.

See video stream captured by camera source plugin and bounding boxes drawn over allowed number of objects identified in that scene on local display device.

Commands:

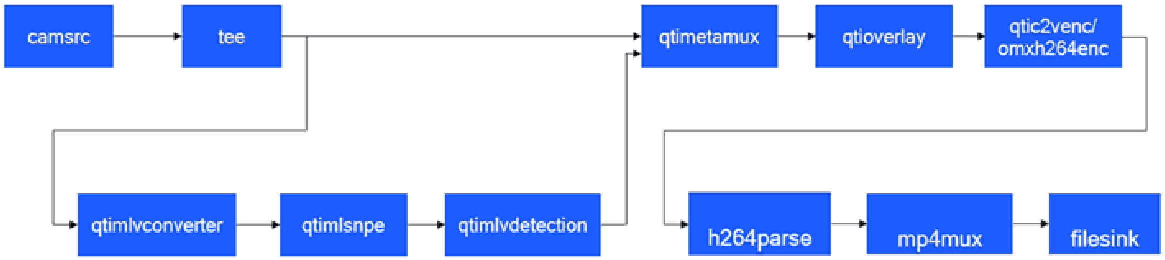
```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtivcomposer name=mixer
```

```
sink_1::dimensions="<1920,1080>" ! queue ! waylandsink sync=false
fullscreen=true split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe
delegate=gpu model=/data/
tf11_public_cnns_cnns_mobilenet_v2_ssd_quant_aware_batch_1_quant.dlc
layers="<Postprocessor/BatchMultiClassNonMaxSuppression>" ! queue !
qtimlvdetection threshold=75.0 results=10 module=ssd-mobilenet labels=/data/
ssd-mobilenet.labels ! video/x-raw,format=BGRA,width=640,height=360 ! queue !
mixer.
```

To stop the use case, press **CTRL + C**.

6.2.4 Single camera stream with object detection and encode with MobileNet v2 SSD

Variant 1: Use qtioverlay plugin to apply detection overlay



The use case is about using mobilenet_v2_ssd model to do the following:

- Identify object in scene from video stream coming through camera source
- Overlay bounding boxes over the detected objects using overlaylib
- Encode this stream as h264 bitstream,later muxing in an MP4 container
- Store the stream as an MP4 file

1. The video stream is collected from camera source plugin and two copies are created
   - One is sent to qtimetamux plugin to retain the video stream
   - Another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that mobilenet_v2_ssd model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the mobilenet_v2_ssd
   b. Modifies the graph for the chosen delegate

   c. Receives the tensor stream on its sinkpad
   d. Executes the inference, collecting outputs from the layers specified to the plugin
   e. Produces tensor stream with the object detection results on its source pad

4. The postprocessing plugin for working on inference results from any object detection model, qtimlvdetection, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvdetection plugin applies the threshold to chose the number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of detection models.

   Here, in this use case, qtimlvdetection loads the ssd-mobilenet submodule and produces results as structures of text and sends them to sinkpad of metamux.
5. The metamux plugin receives video stream and text stream with bounding box results corresponding to video stream on its sinkpads and produces gst buffers with contents of video stream from its sink pad, adding bounding boxes as GstVideoRegionOfInterest from data sinkpad to gst buffer's meta (meta muxing) on its source pad.
6. The qtioverlay plugin receives the muxed stream and overlays the bounding boxes on the VideoFrame using CL and produces gst buffers with overlays in its source pad.
7. The qtic2venc plugin applies parameters to each frame of the video stream it is receiving on its sinkpad and encodes it into bitstream and send it over its sourcepad.
8. H264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates containers format specification buffers
9. The filesink stores the resulting stream in the`/data/video.mp4` file.

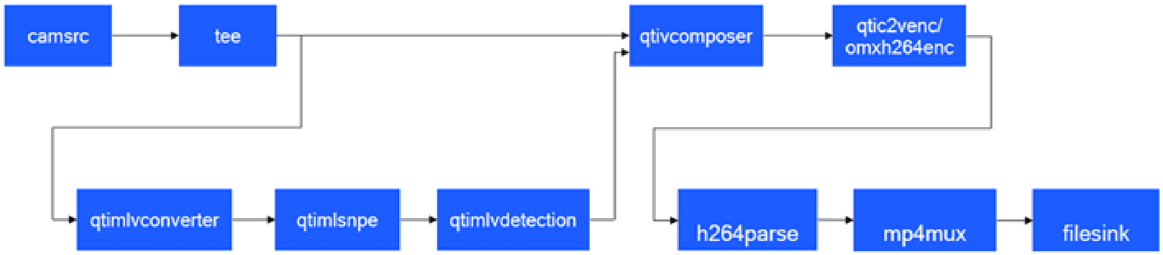
Pull video.mp4 from device and play on a media player application.

Commands:

```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtimetamux name=metamux ! queue ! qtioverlay ! queue !
qtic2venc target-bitrate=6000000 ! h264parse ! queue ! mp4mux ! queue !
filesink location=/data/video.mp4 split. ! queue ! qtimlvconverter ! queue !
qtimlsnpe delegate=gpu model=/data/
tf11_public_cnns_cnns_mobilenet_v2_ssd_quant_aware_batch_1_quant.dlc
layers="<Postprocessor/BatchMultiClassNonMaxSuppression>" ! queue !
qtimlvdetection threshold=75.0 results=10 module=ssd-mobilenet  labels=/data/
ssd-mobilenet.labels ! text/x-raw ! queue ! metamux
```

To stop the use case, press **CTRL + C**.

Variant 2: Use qtivcomposer to mix original frame with detection mask



The use case is about using mobilenet_v2_ssd model to do the following:

- Identify objects in scene from video stream coming through camera source
- Compose bounding boxes over objects detected and original video stream together using qtivcomposer
- Encode this stream as a h264 bitstream, later muxing in an MP4 container
- Store it as an MP4 file

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to qtivcomposer plugin to retain the video stream
   - Another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that mobilenet_v2_ssd model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the mobilenet_v2_ssd model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and collects tensors from specified output layers
   e. Produces tensor stream with the object detection results on its source pad
4. The postprocessing plugin for working on inference results from any object detection model, qtimlvdetection, converts the inference tensors it receives on its sinkpad into formats like video or text that the multimedia plugins can understand later.

   The qtimlvdetection plugin applies the threshold to the chosen number of results the user is looking for. This plugin is capable of loading corresponding modules for a variety of detection models.

   Here, in this use case, qtimlvdetection loads ssd-mobilenet submodule and produces video frames with only bounding boxes that can be overlaid on objects and sends them to sinkpad of qtivcomposer.

5. The qtivcomposer plugin receives original video stream and video stream with bounding boxes on its sinkpads and produces on its sourcepad gst buffers with contents composed of video streams on its sinkpads.
6. The qtic2venc plugin applies parameters to each frame of the video stream it is receiving on its sinkpad and encodes it into bitstream and send it over its sourcepad.
7. H264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates containers format specification buffers
8. Filesink stores the resulting stream in the`/data/video.mp4` file.

Pull video.mp4 from device and play it on a media player application.

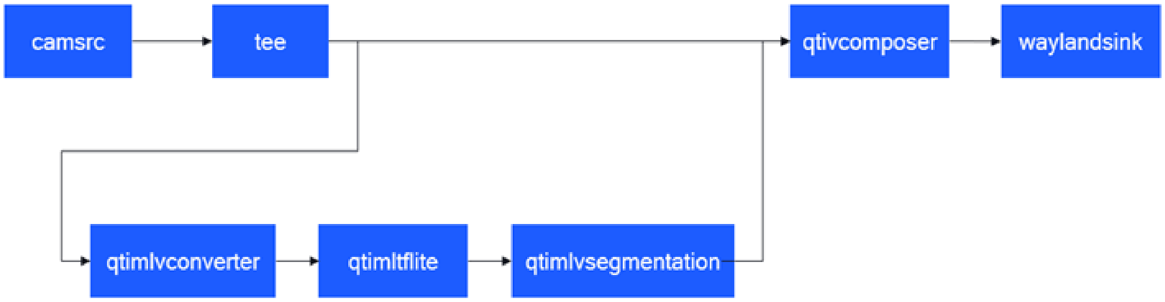
Commands:

```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtivcomposer name=mixer
sink_1::dimensions="<1920,1080>" ! queue ! qtic2venc target-bitrate=6000000 !
h264parse ! queue ! mp4mux ! queue ! filesink location=/data/video.mp4
split. ! queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=gpu model=/data/
tf11_public_cnns_cnns_mobilenet_v2_ssd_quant_aware_batch_1_quant.dlc
layers="<Postprocessor/BatchMultiClassNonMaxSuppression>" ! queue !
qtimlvdetection threshold=75.0 results=10 module=ssd-mobilenet  labels=/data/
ssd-mobilenet.labels ! video/x-raw,format=BGRA,width=640,height=360 ! queue !
mixer
```

To stop the use case, press **CTRL + C**.

6.2.5 Single camera stream with image segmentation and display with DeepLab v3 quantized

Use qtivcomposer to mix original frame with segmentation mask



The use case is about using deeplabv3 model to do the following:

- Identify semantic segmentations in scene from video stream coming through camera source
- Compose the semantics and original video stream together using qtivcomposer
- Display the results on a local display

1. The video stream is collected from camera source plugin and two copies are created:
   - One is sent to qtivcomposer plugin to retain the video stream
   - Another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that deeplab model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the deeplab model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the segmentation results on its source pad
4. The postprocessing plugin for working on inference results from any segmentation model ,qtimlvsegmentation, converts the inference tensors it receives on its sinkpad into video formats that the multimedia plugins can understand later.

   The qtimlvsegmentation plugin produces the semantic segmentations for the frame the user is looking for. This plugin is capable of loading corresponding modules for a variety of segmentation models.

   Here, in this use case, qtimlvsegmentation loads deeplab-argmax submodule and produces video frames with segmentation masks and sends them to sinkpad of qtivcomposer.
5. The qtivcomposer plugin receives original video stream and video stream with segmentation mask on its sinkpads and produces on its sourcepad gst buffers with contents composed of video streams from sinkpads.
6. Waylandsink submits the video stream its receiving on its sinkpad to weston and weston renders the video stream on a local display device.

See video stream captured by camera source plugin and segmentation masks drawn over objects/ components in that scene on local display device.

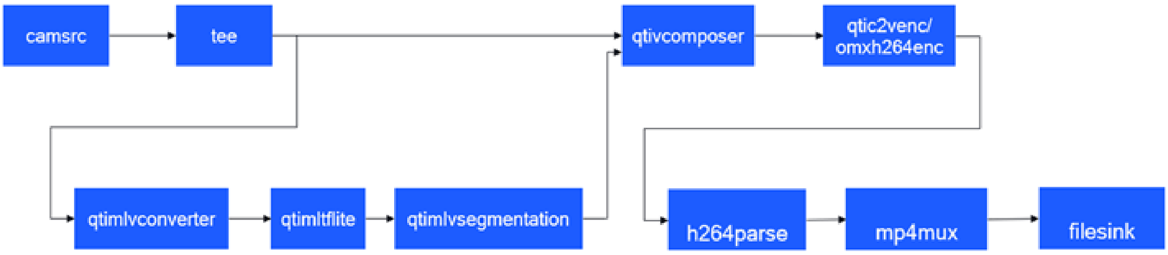
Command:

```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>"
sink_1::alpha=0.5 ! queue ! waylandsink sync=false fullscreen=true split. !
queue ! qtimlvconverter ! queue ! qtimlsnpe delegate=gpu model=/data/
dv3_argmax_int32.dlc ! queue ! qtimlvsegmentation module=deeplab-argmax
labels=/data/dv3-argmax.labels ! video/x-raw,width=256,height=144 ! queue !
mixer
```

To stop the use case, press **CTRL + C**.

6.2.6 Single camera stream with image segmentation and encode with deeplabv3_ quantized

Use qtivcomposer to mix original frame with segmentation mask



The use case is about using deeplab model to do the following:

- Identify semantic segmentations in scene from video stream coming through camera source
- Compose the semantics and original video stream together using qtivcomposer
- Encode this stream as h264 bitstream, later muxing in an MP4 container
- Store it as a MP4 file

1. The video stream is collected from camera source plugin and two copies are created
   - One is sent to qtivcomposer plugin to retain the video stream
   - Another is sent to ML inferencing pipeline
2. The preprocessing plugin, qtimlvconverter, does the following:
   a. Receives the video stream on its sink pad
   b. Does the preprocessing like color conversion, scaling down/up, normalization on the stream data when model expects floating point values as input
   c. Converts the video stream to a tensor stream on its source pad that deeplab model can use for inferencing in later part of pipeline
3. The ML inferencing plugin for SNPE runtime, qtimlsnpe, does the following:
   a. Loads the deeplab model
   b. Modifies the graph for the chosen delegate
   c. Receives the tensor stream on its sinkpad
   d. Executes the inference and produces tensor stream with the segmentation results on its source pad
4. The postprocessing plugin for working on inference results from any segmentation model ,qtimlvsegmentation, converts the inference tensors it receives on its sinkpad into video formats that the multimedia plugins can understand later.

   The qtimlvsegmentation plugin produces the semantic segmentations for the frame the user is looking for. This plugin is capable of loading corresponding modules for a variety of segmentation models.

Here, in this use case, qtimlvsegmentation loads deeplab-argmax submodule and produces video frames with segmentation masks and sends them to sinkpad of qtivcomposer.

5. The qtivcomposer plugin receives original video stream and video stream with segmentation mask on its sinkpads and produces on its sourcepad gst buffers with contents composed of video streams from sinkpads.
6. The qtic2venc plugin applies parameters to each frame of the video stream it's receiving on its sinkpad and encodes it into bitstream and send it over its sourcepad.
7. H264parse adds additional information corresponding to the bitstream to gstreamer buffer meta and mp4mux plugin receives these buffers and creates containers format specification buffers
8. The filesink stores the resulting stream in the`/data/video.mp4` file.

Pull video.mp4 from device and play it on a media player application.

Command:

```
gst-launch-1.0 -e --gst-debug=2 qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! tee
name=split ! queue ! qtivcomposer name=mixer sink_1::dimensions="<1920,1080>"
sink_1::alpha=0.5 ! queue ! qtic2venc target-bitrate=6000000 ! h264parse !
queue ! mp4mux ! queue ! filesink location=/data/video.mp4 split. ! queue !
qtimlvconverter ! queue ! qtimlsnpe delegate=gpu model=/data/
dv3_argmax_int32.dlc ! queue ! qtimlvsegmentation module=deeplab-argmax
labels=/data/dv3-argmax.labels ! video/x-raw,width=256,height=144 ! queue !
mixer.
```

To stop the use case, press **CTRL + C**.

7 Example applications

7.1 gst-activate-deactivate-streams-runtime

The application performs link/unlink of the streams without reconfiguration of the camera and ensures that the other streams do not have any gaps.

- It demonstrates the ability of qtiqmmfsrc to activate/deactivate its pads runtime without the need of any reconfiguration or gap.
- It creates three video streams with different resolutions and activates/deactivates them in different order and while in different states.

Usage

```
# Since output=Display provided, run required commands to connect to display
gst-activate-deactivate-streams-runtime-example --usecase=Full --
output=Display
```

Table 7-1 User menu of gst-activate-deactivate-streams-runtime

Option	Accepted values	Description
--help or -h	–	Help
--use case or -u	Basic (Default)	More straightforward version to test
	Full	More number of iterations of linking/unlinking streams

Table 7-1 User menu of gst-activate-deactivate-streams-runtime (cont.)

Option	Accepted values	Description
--output or -o	File (Default)	Pipeline used internally : `qtiqmmfsrc name=qmmf qmmf.video_0 ! capsfilter caps="video/x-raw\(memory:GBM\), format=NV12, width=w0, height=h0, framerate=30/1" ! qtic2venc target-bitrate=6000000 ! h264parse ! mp4mux ! filesink location=/data/video_0.mp4 qmmf.video_1 ! capsfilter caps="video/x-raw\(memory:GBM\), format=NV12, width=w1, height=h1, framerate=30/1" ! qtic2venc target-bitrate=6000000 ! h264parse ! mp4mux ! filesink location=/data/video_1.mp4 qmmf.video_2 ! capsfilter caps="video/x-raw\(memory:GBM \), format=NV12, width=w2, height=h2, framerate=30/1" ! qtic2venc target-bitrate=6000000 ! h264parse ! mp4mux ! filesink location=/data/video_2.mp4`
	Display	Pipeline used internally: `qtiqmmfsrc name=qmmf qmmf.video_0 ! capsfilter caps="video/x-raw\(memory:GBM\), format=NV12, width=w0, height=h0, framerate=30/1" ! waylandsink x=x0 y=y0 width=640 height=480 async=TRUE enable-last-sample=FALSE qmmf.video_1 ! capsfilter caps="video/x-raw\(memory:GBM \), format=NV12, width=w1, height=h1, framerate=30/1" ! waylandsink x=x1 y=y1 width=640 height=480 async=TRUE enable-last-sample=FALSE qmmf.video_2 ! capsfilter caps="video/x-raw\(memory:GBM\), format=NV12, width=w2, height=h2, framerate=30/1" ! waylandsink x=x2 y=y2 width=640 height=480 async=TRUE enable-last-sample=FALSE`

7.2 gst-add-remove-streams-runtime

The application performs link/unlink of the streams but the reconfiguration of the camera happens every time a stream is being linked (a 200 ms gap observed in the other streams).

- It demonstrates the ability of qtiqmmfsrc to add/remove streams runtime with camera reconfiguration.
- It creates three video streams with different resolutions and adds/removes them in different order and while in different states.
- It is different from gst-activate-deactivate-streams-runtime app as here we're requesting and releasing the pads each time, whereas in activate-deactivate-app we are just activating and deactivating an already requested pad, and releasing only in the end.

Usage

```
# Run required commands to connect to display
gst-add-remove-streams-runtime-example
```

Table 7-2 User menu of gst-add-remove-streams-runtime

Use case	Steps to execute the Use case.	Observations
Add/remove the streams runtime with camera reconfiguration	export XDG_RUNTIME_DIR=/run/user/root && gst-add-remove-streams-runtime-example	The streams appears on the screen. The application performs streams start/stop each 5 seconds and it is visible on the screen. Here is the work flow of the application: ▪ Create 1080p stream (up-left corner of the display) ▪ Create 720p stream (up-right corner of the display) ▪ Create 480p stream (bottom-left corner of the display) ▪ Remove 1080p stream ▪ Remove 720p stream ▪ Pause the pipeline ▪ Create 1080p stream in paused state (up-left corner of the display) ▪ Resume the pipeline ▪ Remove 1080p stream ▪ Remove all streams ▪ End of execution Note that in this use case the new stream affects the others with about 200 ms gap in the video.

7.3 gst-add-streams-as-bundle-example

This sample application demonstrates the ability to configure streams as a bundle instead of configuring each stream one after another.

- It helps reduce the delay caused by configuring streams each time a new stream is added
- It creates a 1080p stream and sets the pipeline in playing state. After few seconds of running, the pipeline is moved to ready and a 720p and 480p stream is added and set to playing state. When the playing state is set, all the 3 streams are configured together.
- The two new streams added, should be created in a bundle without a delay between them.

Usage

Since output=Display provided, run required commands to connect to display

```
gst-add-streams-as-bundle-example --output=Display
```

Table 7-3 User menu of gst-add-streams-as-bundle-example

Option	Accepted values	Description
--help or -h		Help
-o <value> or --output=value	"File" or "Display"	What output to use

7.4 gst-appsink-example

It demonstrates the use of appsink plugin to enable an application to catch incoming buffers in the pipeline.

- Whenever the appsink emits new-sample signal, i.e. a new sample is available in the pipeline, the app extracts the buffer from the sample and prints "Received a buffer, doing some processing..." on the screen.
- The pipeline used by the app is

```
qtiqmmfsrc name=camera !
        video/x-raw, format=NV12, width=1920, height=1080,
framerate=30/1 ! queue ! appsink name =
        sink emit-signals=true
```

Usage: `gst-appsink-example`

7.5 gst-appsink-raw-plus-yuv

This application connects the camera with two appsink streams - one for yuv frame capture and one for raw frame capture.

- Whenever the appsinks emit new-sample signal, i.e. a new sample is available, the buffer is saved to device storage in the following files:
  - /data/frame_<frame_no.>_w_<width>_h_<height>_stride_<stride>_scanline_<offset_data>.raw
  - /data/frame_<frame_no.>_w_<width>_h_<height>_stride_<stride>_scanline_<offset_data>.yuv
- The pipeline used by the app is

```
qtiqmmfsrc name=camera !
        capsfilter caps="video/x-raw, format=NV12, width=1280, height=720,
framerate=30/1" ! queue
        ! appsink name = yuv_appsink emit-signals=1 camera. ! capsfilter
caps="video/x-bayer,
        format=rggb, bpp=(string)10, width=w, height=h, framerate=30/1" !
queue ! appsink name =
        raw_appsink emit-signals=1.
```

Usage

```
gst-appsink-raw-plus-yuv-example
```

7.6 gst-camera-capture-example

This application demonstrates the burst snapshot capability with AE bracketing support.

- It creates a 720p video stream which is sent to a display and a 1080p JPEG snapshot stream. When in playing state, we fetch the Image metadata (AE compensation range) and set the values for AE compensation with minimum to maximum with a step for each image based on number of images requested for capture
- Once we receive images after we set the AE compensation, we switch to still video capture and run the pipeline for another 15 seconds

Usage

Run required commands to connect to display:

```
export XDG_RUNTIME_DIR=/run/user/root && gst-camera-capture-example
```

The captured images are saved at`/data/frame_*.jpg` on the device.

7.7 gst-camera-burst-intervalcapture-example

This application runs burst snapshot in equal interval time gap.

When running a snapshot, display will be paused and after snapshot, display will resume. During burst snapshot, snapshot does not need to wait for AE converged if scene/brightness changes.

Burst snapshot is needed on the 4 options (5 Pics in 1 sec, 10 pictures in 1 second, 15 pictures in 3 seconds, 30 pictures in 3 seconds). The app only requests when it needs the JPEG file, it does not request any frame between images. It sends request every 100/200 ms.

For example: 5 Pics in 1 sec:

1. 0ms: 1st process_request and save JPEG
2. (1st Snapshot) -> no process_request from app
3. 200ms: 2nd process_request and save JPEG
4. (2nd Snapshot) -> no process_request from app ->
5. …
6. 800ms: 5th process_request and save JPEG (5th Snapshot)
7. finish.

Usage

As a prerequisite, Weston should be running for display use cases:

```
mount -o remount,rw / && export XDG_RUNTIME_DIR=/run/user/root && export
WAYLAND_DISPLAY=wayland-1
```

camxoverridesettings.txt:

```
enableFeature2CTS=1
```

Run the required commands to connect to display (if using -d option):

Table 7-4 User menu of gst-camera-burst-intervalcapture-example

Option	Description
-w	Image width of stream
-h	Image height of stream
-a	Preview width of stream
-b	Preview height of stream
-d	Rounds of burst snapshot
-p	Preview output type: ■ 0 – AVC ■ 1 – Display
-c	Capture format type: ■ 0 – JPEG ■ 1 – YUV ■ 2 – BAYER ■ 3 – JPEG + BAYER
-r	Capture interval capture requires: ■ 0 – 5 images in 1 second ■ 1 – 10 images in 1 second ■ 2 – 15 images in 3 seconds ■ 3 - 30 images in 3 seconds

Table 7-5 Test cases of gst-camera-burst-intervalcapture-example

Test case	Command	Observations
Capture 5 Pictures in 1 seconds	`gst-camera-burst-intervalcapture-example -r 0`	1. Wayland with size 960x720 runs for 10 seconds, then pauses at the last frame. 2. Capturing begins and Wayland send capture signal every 200 ms for 5 times. 3. Wayland resumes and runs for 10 seconds. The program gets EOS and finishes by itself. 4. The output of 5 JPEG images is saved at`/data/frame_*`.
Capture 10 Pictures in 1 seconds	`gst-camera-burst-intervalcapture-example -r 1`	1. Wayland with size 960x720 runs for 10 seconds, then pauses at the last frame. 2. Capturing begins and Wayland sends capture signal every 100 ms for 10 times. 3. Wayland resumes and runs for 10 seconds.

Table 7-5 Test cases of gst-camera-burst-intervalcapture-example (cont.)

Test case	Command	Observations
		4. The program gets EOS and finishes by itself. 5. The output of 10 JPEG images is saved at`/data/frame_*`.
Capture 15 Pictures in 3 seconds	`gst-camera-burst-intervalcapture-example -r 2`	1. Wayland with size 960x720 runs for 10 seconds, then will pause at the last frame. 2. Capturing begins and Wayland sends capture signal every 200 ms for 15 times. 3. Wayland resumes and runs for 10 seconds. 4. The program gets EOS and finishes by itself. 5. The output of 15 JPEG images is saved at`/data/frame_*`.
Capture 30 Pictures in 3 seconds	`gst-camera-burst-intervalcapture-example -r 3`	1. Wayland with size 960x720 runs for 10 seconds, then pauses at the last frame. 2. Capturing begins and Wayland sends capture signal every 100 ms for 30 times. 3. Wayland resumes and runs 10 seconds. 4. The program gets EOS and finishes by itself. 5. The output of 30 JPEG images is saved at`/data/frame_*`.
Capture 30 Pictures in 3 seconds for 3 rounds	`gst-camera-burst-intervalcapture-example -r 3 -d 3`	1. Wayland with size 960x720 runs for 10 seconds, then will pauses at the last frame. 2. Capturing begins and Wayland sends capture signal every 100 ms for 30 times. 3. Wayland resumes and runs for 10 seconds. 4. The program gets EOS and finishes by itself. 5. The output of 30 JPEG images is saved at`/data/frame_*`.

Table 7-5 Test cases of gst-camera-burst-intervalcapture-example (cont.)

Test case	Command	Observations
		This repeats thrice.
Capture 30 Pictures in 3 seconds and set preview size 1280x720	`gst-camera-burst-intervalcapture-example -r 3 -a 1280 -b 720`	Wayland with size 1280x720 runs for 10 seconds, then pauses at the last frame. Capturing begins and Wayland sends capture signal every 100 ms for 30 times. Wayland resumes and runs for 10 seconds. The program gets EOS and finishes by itself. The output of 30 JPEG images are saved at`/data/frame_*`.

7.8 gst-camera-metadata-example

The application starts with a menu that enables you to choose between the three properties of camera plugin, that is, video-metadata, static-metadata and image-metadata.

On choosing one, it presents a menu with options:

1. List all available tags. This will list all the tags in the meta on console.
2. Dump all tags values in a file. This will print all the tags with their values in a file.
3. Dump custom tags values in a file. This will print the values of tags specified in the user-provided config file in a file.
4. Get a tag. This will allow the user to get value of a particular tag.
5. Set a tag (only for video-metadata).This will allow the user to set value of a particular tag.
   - It also has options to dump the values of tags coming in result-metadata and urgent-metadata signals in a file.
   - It also allows registering to new_sample signal of appsink and dumping camera timestamp from each frame in a file.
   - It also provides an option to see the live preview on display so that the user can see the effect live while setting a particular tag.
     - The pipeline used by the app is

       ```
       qtiqmmfsrc name=camera ! video/x-raw\(memory:GBM\), format=NV12,
       width=1280, height=720, framerate=30/1 ! queue ! appsink name = sink
       emit-signals=true for normal usecase, and qtiqmmfsrc name=camera
       camera.video_0 ! video/x-raw (memory:GBM), format=NV12, width=1280,
       height=720, framerate=30/1 ! queue ! appsink name=sink emit-
       signals=true async=false enable-last-sample=false camera.video_1 !
       video/x-raw (memory:GBM), format=NV12, width=1280, height=720,
       framerate=30/1 ! queue ! waylandsink fullscreen=true for display
       usecase.
       ```

Usage

```
gst-camera-metadata-example
```

Table 7-6 User menu of gst-camera-metadata-example

Option	Description
--help or -h	Help
--display or -d	Enable display output
--timestamps-location or -t	File in which camera timestamps will be recorded
--urgent-meta-location or -u	File in which urgent-metadata tags' values will be recorded
--result-meta-location or -r	File in which result-metadata tags' values will be recorded

Use case: Read and print urgent metadata from test application

- Sample application command:

  ```
  gst-camera-metadata-example -u /data/urg.log
  ```

- Example observation:

  Pull the `/data/urg.log` file and check if urgent metadata appears in the file. For example:

  ```
  ------------------------------ Android tags ---------------------------
  ------ SECTION ----- --------- TAG ---------- --- VALUE ----
   android.control afMode 0
   android.control awbMode 1
   android.control aeState 1
   android.control afState 0
   android.control awbState 0
   android.request frameCount 0
   android.request id 0
  ---------------------------------------------------------------------------
  -

  ------------------------------ Android tags ---------------------------
  ------ SECTION ----- --------- TAG ---------- --- VALUE ----
   android.control afMode 0
   android.control awbMode 1
  ...
  ...
  ...
   ...
  ```

7.9 gst-camera-switch-appsrc-example

This application uses two cameras of the device and switches between them without changing the state of the pipeline and without any interruption/gap with only one camera being active at any point of time.

- The application creates three pipelines. Two for camera 0 and camera 1 connected to appsink and one with appsrc.
- The stream from main camera will appear on the screen first and the app will perform switching between preview of the two cameras after every 5 seconds while the pipeline is in playing state. To exit, press Ctrl+C.
- When switching only one camera is active at the moment and this is achieved by using a buffer pool which is activated when we stop the camera pipeline and until all buffers are returned to the qmmf-sdk.
- The buffer pool keeps the appsrc pipeline active (which has the encoder/display instance) until the buffers start coming from the next camera.
- By default, it will output the video data to filesink and save the video file to*/data/mux.mp4*.

Usage

Run required commands to connect to display (if using -d option):

```
gst-camera-switch-appsrc-example
```

Table 7-7 User menu of gst-camera-switch-appsrc-example

Option	Description
--help or -h	Help
--display or -d	Enable display output (if not selected, the content is encoded and saved in a file)
--camera0 or -m	ID of camera0
--camera1 or -s	ID of camera1
--width or -w	Output width (for both cameras)
--height or -h	Output height (for both cameras)
--delay or -l	Camera switch delay (in seconds) - Default: 5 seconds

Table 7-8 Use case on gst-camera-switch-appsrc-example

Test case	command	Observations
Camera switch dynamically via link/ unlink encode	gst-camera-switch-appsrc-example -m 0 -s 1	The stream from main cameras will encode to local mp4 file. The application will perform switching between recording of the both cameras each 5 seconds. Here is the work flow of the application: ■ Create 720p stream from camera 0 and record to mp4 file ■ After 5 seconds create a new 720p stream from camera 1 ■ Unlink the old and link the new stream

Table 7-8 Use case on gst-camera-switch-appsrc-example (cont.)

Test case	command	Observations
		■ Set the old stream the NULL and remove it ■ Repeat until press CTRL+C
Camera switch dynamically via link/ unlink display	gst-camera-switch-appsrc-example -m 0 -s 1 -d	The stream from main cameras will appears on the screen. The application will perform switching between preview of the both cameras each 5 seconds. Here is the work flow of the application: ■ Create 720p stream from camera 0 and link it to the waylandsink ■ After 5 seconds create a new 720p stream from camera 1 ■ Unlink the old and link the new stream ■ Set the old stream the NULL and remove it ■ Repeat until press CTRL+C

7.10 gst-camera-switch-example

This application uses two cameras of the device and switches between them without changing the state of the pipeline and without any interruption/gap.

- The stream from main camera will appear on the screen first and the app will perform switching between preview of the two cameras after every 5 seconds while the pipeline is in playing state. To exit, press **CTRL + C**.
- For switching, new instance of *qtiqmmfsrc* element is created with '*camera*' property 0 or 1 and linked to the pipeline after unlinking and removing the old one.
- By default, it will output the video data to filesink and save the video file to`/data/mux.mp4`.

Usage

Run required commands to connect to display (if using -d option)

```
gst-camera-switch-example -d
```

Table 7-9 User menu of gst-camera-switch-example

Option	Description
--help or -h	Help
-d	Enable display output

7.11 gst-depth-assist-autofocus-example

This application is used to enable the depth data which is used to assist the auto focus.

It implements one thread to monitor the keyboard to get the input parameters as the dummy depth data, and then update these data to the other thread which simulates the depth sensor to continuously send the depth data to camera service via vendor tag.

Usage

```
gst-depth-assist-autofocus-example
```

Table 7-10 User Menu (Runtime)

Option	Accepted values	Description
Depth tof data flag	0 - disable, 1 - enable	Enable/Disable Depth tof data
Distance value (in millimeters)	100 - 10000	Set the distance value
Distance confidence level	0 - 2	Set the distance confidence level
Depth distance near limitation	100 - 10000	Set depth distance min value in millimeter
Depth distance far limitation	100 - 10000	Set depth distance max value in millimeters

7.12 gst-single-4k-example

This application creates one 4K stream (Default resolution), encodes, muxes and saves it in an MP4 file. The user can also configure the video with desired width and height.

Usage

```
 gst-single-4k-example
```

The encoded stream is saved at`/data/mux.mp4` on the device

Table 7-11 User menu of gst-single-4k-example

Option	Description
--help or -h	Help
--width or -w	Image width (Default=3840)
--height or -h	Image height (Default=2160)

7.13 gst-three-1080p-example

This application creates three 1080p streams (default resolution). One stream is sent to display, the second is encoded (H264) to MP4, and the third is streamed via RTSP.

Usage

Run required commands to connect to display:

1. On the host, run:

   ```
   adb forward tcp:8900 tcp:8900
   ```

2. In one device console, run:

   ```
   gst-rtsp-server -p 8900 -m /live "( udpsrc name=pay0 port=8554 caps=
   \"application/x-rtp,media=video,clock-rate=90000,encoding-
   name=H264,payload=96\" )"
   ```

3. In another console, run:

   ```
   export XDG_RUNTIME_DIR=/run/user/root && gst-three-1080p-example
   ```

4. Open VLC application on PC and go to **Media** > **Open Network Stream**and enter `rtsp://127.0.0.1:8900` for network URL to view the RTSP stream.

   The encoded stream is saved at`/data/mux.mp4` on the device.

Table 7-12 User menu of gst-three-1080p-example

Option	Description
--help or -h	Help
--width or -w	Image width (Default=1920) for all three streams
--height or -h	Image height (Default=1080) for all three streams

7.14 gst-two-1080p-example

This application creates two 1080p streams. One stream is saved to a YUV file and one is encoded (H264) to MP4.

Usage

```
gst-two-1080p-example
```

The encoded stream is saved at`/data/mux.mp4` on the device. The yuv bitstream is saved at`/data/vid.yuv` on the device

Table 7-13 User menu of gst-two-1080p-example

Option	Description
--help or -h	Help
--width or -w	Image width (Default=1920) for both streams
--height or -h	Image height (Default=1080) for both streams

7.15 gst-weston-composition

This application shows composition of video previews from different sources using waylandsink. It performs zoom in and zoom out of each pipeline's preview in order after every 3 sec.

Usage

Run required commands to connect to display. It's important to name each waylandsink plugin 'wayland', otherwise it generates an 'Invalid plugin name' error.

```
gst-weston-composition-example -s "qtiqmmfsrc camera=0 ! queue ! waylandsink
name=wayland x=0 y=0 width=640 height=360" "qtiqmmfsrc camera=1 ! queue !
waylandsink name=wayland x=640 y=0 width=640 height=360"
```

Table 7-14 User menu of gst-weston-composition

Option	Description
--help or -h	Help
--source or -s *(must be specified)*	▪ Waylandsink pipelines that are going to be used. ▪ Each pipeline must be enclosed within quotes. ▪ Maximum of 4 is allowed.

7.16 gst-tflite-yolo-ssd-display-example

This application demonstrates running of the TFLite Detection model using the Qualcomm ML GStreamer plugins.

The use case decodes an input file, runs detection inferencing on the stream, and then overlays the detection data onto the decoded stream, which is displayed on the monitor.

Usage

1. Run the required commands to connect to display.
2. Push the necessary detection model and label file to`/data`
   - Yolov5m-320x320-int8.tflite
   - yolov5s-320x320-int8.tflite
   - ssd-mobilenet_v1_1.tflite
   - ssd-mobilenet.labels
   - yolov5m.labels

```
gst-tflite-yolo-ssd-display-example
```

Table 7-15 User menu of gst-tflite-posenet-display-example

Option	Description
--helpor -h	Help
-por --postproc=<model>	0– yolov5m,1 – yolov5s, 2 - ssd-mobilenet

7.17 gst-tflite-posenet-display-example

This application demonstrates running of the TFLite PoseNet model using the Qualcomm ML GStreamer plugins.

The use case decodes an input file, runs PoseNet inferencing on the stream and then overlays the pose data onto the decoded stream, which is displayed onto the monitor.

Usage

1. Run the required commands to connect to display.
2. Push the necessary PoseNet model and label file to `/data/posenet_mobilenet_v1_075_481_641_quant.tflite` and `/data/posenet.labels` respectively:

```
gst-tflite-posenet-display-example -d 0 -i <input mp4 file>
```

Table 7-16 User menu of gst- tflite-posenet-display-example

Option	Description
--help or -h	Help
-d or --decoder=<decoder>	0 - qtic2vdec, 1 - omxh264dec (It's only qtic2vdec for QCS8550)
-i, --input_file=<file>	Input filename

8 GStreamer Daemon

GStreamer daemon (GStD) is a process that abstracts much of the complexity of writing custom GStreamer applications. It also factors out the boilerplate code that required to write applications from scratch.

GStD runs independently and exposes a public interface over a variety of IPC mechanisms for other processes to communicate with and control the daemon. The design behind Gstd follows a model-view-controller (MVC) architecture.

- GstD provides the core and IPC endpoint.
  a. The GstD Core holds the current state of the GStreamer daemon, such as pipelines created along with their respective states, elements in each pipeline, properties in each element, and so on.
  b. The *IPC* allows client applications to alter the state of the Gstd core using an interprocess communication mechanism, that is, TCP, HTTP, DBus, sockets, etc. Custom applications use the IPC to modify the state of the Gstd core like create a new pipeline, set the state to play, or change an element property while the pipeline is running.
- The custom application provides view and client logic.
  a. The *View* provides visual feedback of the Gstd core state to the user. This is typically a graphical user interface, a web page, a cmdline application, etc. The user will see the view, which in turn is updated by the model.
  b. The *client logic* provides the application's custom functionality. The user will use the IPC/ controller to modify the Gstd core/model and will receive feedback via the GUI/view.

Distributed with the project, there is a GstdClient: a simple, cmdline based application that talks via TCP with GStreamer Daemon. GstdClient is like *gst-launch* as you can create and start pipelines. Unlike gst-launch, GstdClient allow you to have several pipelines active with the ability to control the pipeline and receive feedback once the pipeline has been created. The GStreamer Daemon project packages the core, the cmdline client and IPC interfaces.

For more information, see https://developer.ridgerun.com/wiki/index.php/GStreamer_Daemon

Run GsTD

1. Start GStreamer Daemon.

```
gstd
# This will start the process in the background. Using 'gstd &' is not
required becauses gstd daemonizes itself by default.

# To run gstd in foreground
gstd -D
```

```
# To view a list of all supported options,
gstd --help-all
```

2. Start GStreamer Client.

```
gst-client
# gstd prompt will appear like,
# gstd>
# In the prompt, we can interact with the server creating/playing
pipelines, etc.

# To list all the possible commands
gstd> help

# Alternatively, instead of opening gstd prompt, can give the command as
argument to gst-client
gst-client list-pipelines
```

8.1 GStreamer Daemon features

Low-level CRUD

The Create, Read, Update, Delete (CRUD) API is important in applications where the human doesn't have the control, this API allows creating custom commands using low-level.

CRUD syntax is usually of the form *command <URI> <description if applied>*. For example, *create / pipelines p1 videotestsrc ! autovideosink*

Different branches from the tree below can be taken to build an URI

Response format

Gstd receives the commands and prints the answer in raw JSON. The answer consists of three parts:

- *code:*Number value that indicates the status of the command.
- *description:*Description of the status code.
- *response:*Body of the command executed.

Interacting with pipelines

Table 8-1 Commands used to interact with pipelines are prefixed with *pipeline_<action>*

High level command	Low level CRUD	Description
`pipeline_create <name> <description>`	`create /pipelines <name> <description>`	Creates a new pipeline named after name using the description.
`pipeline_play <name>`	`update /pipelines/<name>/state playing`	Puts the pipeline named 'name' in the PLAYING state.
`pipeline_pause <name>`	`update /pipelines/<name>/state paused`	Puts the pipeline named 'name' in the PAUSED state.
`pipeline_stop <name>`	`update /pipelines/<name>/state null`	Puts the pipeline named 'name' in the NULL state.
`pipeline_delete <name>`	`delete /pipelines/<name>`	Deletes the pipeline named 'name', stopping it first if necessary.
`list_pipelines`	`read /pipelines`	List the pipelines in the session.
Refcount commands provide an alternative to interact with pipelines as though they had a reference counter. It's useful to ensure thread safety when multiple processes share a single pipeline.		
pipeline_create_ref <name> <description>	–	Creates a new pipeline or increases the create refcount if the pipeline already exists. *(If we'd used basic commands for pipeline creation by two processes, it could've caused a state where same pipeline might have been created twice. Using this command, it will just increase the refcount instead of creating a duplicate pipeline.)*
pipeline_delete_ref <name>	–	Deletes the pipeline or decreases the create refcount if refcount > 1.
pipeline_play_ref <name>	–	Puts the pipeline in the PLAYING state or increases the play refcount if the pipeline is already playing.
pipeline_stop_ref <name>	–	Puts the pipeline in the NULL state or decreases the play refcount if refcount > 1.

Modifying element properties

Table 8-2 Commands used to interact with pipeline elements are prefixed with element_<action>

High level command	Low level CRUD	Description
`element_get <pipeline> <element> <property>`	`read /pipelines/<pipeline>/elements/<element>/properties/<property>`	Reads the value of a property of an element in a pipeline.
`element_set <pipeline> <element> <property> <value>`	`update /pipelines/<pipeline>/elements/<element>/properties/<property> <value>`	Updates the value of a property of an element in a pipeline to value.
`list_properties <pipeline> <element>`	`read /pipelines/<pipeline>/elements/<element>/properties`	List the properties of an element in a pipeline.

Sending events

High level command	Low level CRUD	Description
`event_eos <pipeline>`	`create /pipelines/<pipeline>/event_eos`	Send EOS event to the pipeline.
`event_seek <pipe> <rate=1.0> <format=3> <flags=1> <start-type=1> <start=0> <end-type=1> <end=-1>`	`create /pipelines/<name>/event_seek <rate=1.0> <format=3> <flags=1> <start-type=1> <start=0> <end-type=1> <end=-1>`	Sends a seek event to an element.
`event_flush_start <pipeline>`	`create /pipelines/<name>/event flush_start`	Put the pipeline in flushing mode, i.e. flush all queued data in the pipeline for new media play.
`event_flush_stop <pipeline> <reset=true>`	`create /pipelines/<name>/event flush_stop <reset=true>`	Take the pipeline out of flushing mode, to enable the pads to receive new data.

Receiving messages from bus

Table 8-3 Commands used to interact with bus are prefixed with *bus_<action>*

High level command	Description
`bus_read <name>`	Blocks and reads the bus of the pipeline with name 'name'.
`bus_filter <name> <filter>`	Reads and filters the messages according to 'filter' from the pipeline named 'name'. Supported bus messages
`bus_timeout <name> <time in nanoseconds>`	Waits for 'time' to read the message defined with 'filter' and then unblocks.

For example:

```
# In gstd prompt
# Create the pipeline that generate an error
pipeline_create p filesrc location=/tmp/test.avi ! identity error-
after=2000 ! avidemux ! avdec_mpeg4 ! fpsdisplaysink

# Filter message error and eos
bus_filter p error+eos

# wait 100s to read message error/eos, if not, returns
bus_timeout p 100000000000

# Play the pipeline
pipeline_play p

# Waiting until bus read a message error or eos
```

Receiving Signals

Table 8-4 Commands used to interact with signals are prefixed with *signal_<action>*

High level command	Low level CRUD	Description
list_signals <pipeline> <element>	read /pipelines/<pipeline>/elements/<element>/signals	List the signals of an element in a pipeline.
signal_connect <pipeline> <element> <signal>	read /pipelines/<pipeline>/elements/<element>/signals/<signal>/ callback	Waits for the signal of the element in pipeline to occurs and returns its arguments.

Table 8-4 Commands used to interact with signals are prefixed with *signal_<action>* (cont.)

High level command	Low level CRUD	Description
signal_timeout <pipeline> <element> <signal> <value>	update /pipelines/<pipeline>/elements/<element>/signals/<signal>/timeout <value>	Change signal wait timeout to value
signal_disconnect <pipeline> <element> <signal>	read /pipelines/<pipeline>/elements/<element>/signals/<signal>/disconnect	Disconnect from signal of the element in pipeline.

Enable the debug subsystem

Table 8-5 Commands used to deal with debug subsystem are prefixed with *debug_<action>*

High level command	Low level CRUD	Description
debug_enable <true/false>	update /debug/enable <true/false>	Enable debug of the pipeline
debug_threshold <threshold>	update /debug/threshold <threshold>	Set the level of debug
debug_color <true/false>	update /debug/color <true/false>	Enable the color in debug

Example

```
# In gstd prompt
# Enable debug
debug_enable true

# Set the level of debug
debug_threshold qti*:6

# Disable color in logs
debug_color false

# Create pipeline
pipeline_create p qtiqmmfsrc ! waylandsink

# Play the pipeline
pipeline_play p
```

By default, Gstd redirects its GStreamer logs to a folder named gstd, created at*/usr/local/var/log*. There, we can find "*gstd.log*" and "*gst.log*":

- */usr/local/var/log/gstd/gstd.log*: Contains GStreamer Daemon debug messages.
- */usr/local/var/log/gstd/gst.log*: Contains all the Gstreamer log messages (including debug).

Thedebug log paths can be changed as shown:

```
# While launching gstd:
gstd --gstd-log-filename /data/gstd.log --gst-log-filename /data/gst.log
```

8.2 GsTD examples

Camera Preview - Single stream live capture YUV

```
# To run gstd usecases in permissive mode
adb shell setenforce 0
adb shell

# Start GStreamer Daemon
gstd

# Create the pipeline
gst-client pipeline_create cam_preview qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink
fullscreen=true async=true sync=false

# Run commands to connect to display, and make sure the variables are set
correctly in /etc/default/gstd
# Play the pipeline
gst-client pipeline_play cam_preview

# Stop the pipeline whenever done
gst-client pipeline_stop cam_preview

# Delete the pipeline
gst-client pipeline_delete cam_preview
```

Camera + Video Encode - Three 1080p AVC streams

```
# To run gstd usecases in permissive mode
adb shell setenforce 0
adb shell

# Start GStreamer Daemon
gstd

# Create the pipeline
gst-client pipeline_create cam_enc qtiqmmfsrc name=camsrc ! video/x-raw\
(memory:GBM\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue !
```

```
qtic2venc target-bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20
max-quant-i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-
frames=20 ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/
mux1.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc
target-bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20 max-quant-
i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux2.mp4" \
camsrc. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! queue ! qtic2venc
target-bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20 max-quant-
i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20 !
queue ! h264parse ! mp4mux ! queue ! filesink location="/data/mux3.mp4"

# Play the pipeline
gst-client pipeline_play cam_enc

# Filter message eos
gst-client bus_filter cam_enc eos

# When done recording, send the EOS event to avoid corruption. Without it,
some header information in the MP4 standard wouldn't be updated and the file
wouldn't be able to be reproduced.
gst-client event_eos cam_enc

# Wait until bus reads the message eos
gst-client bus_read cam_enc

# Delete the pipeline when eos received
gst-client pipeline_delete cam_enc
```

Multi camera use case - Client1, main camera: 4k at 30 AVC MP4, 1080p at 30 YUV preview. Client2, secondary camera: 720p at 30 AVC, 720p at 30 YUV

```
# To run gstd usecases in permissive mode
adb shell setenforce 0
adb shell

# Start GStreamer Daemon
gstd

# Create the pipeline
gst-client pipeline_create multicam qtiqmmfsrc name=camsrc_0 ! video/x-raw\
(memory:GBM\),format=NV12,width=3840,height=2160,framerate=30/1 ! queue !
qtic2venc target-bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20
max-quant-i-frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-
frames=20 ! queue ! h264parse ! mp4mux ! queue ! filesink location="/data/
```

```
mux1.mp4" \
camsrc_0. ! video/x-raw\(memory:GBM
\),format=NV12,width=1920,height=1080,framerate=30/1 ! waylandsink x=0 y=0
width=500 height=400 async=true sync=false \
qtiqmmfsrc name=camsrc_1 camera=1 ! video/x-raw\(memory:GBM\), format=NV12,
width=1280, height=720, framerate=30/1 ! queue ! qtic2venc target-
bitrate=6000000 min-quant-i-frames=20 min-quant-p-frames=20 max-quant-i-
frames=30 max-quant-p-frames=30 quant-i-frames=20 quant-p-frames=20 ! queue !
h264parse ! mp4mux ! queue ! filesink location="/data/mux2.mp4" \
camsrc_1. ! video/x-raw\(memory:GBM\), format=NV12, width=1280, height=720,
framerate=30/1 ! waylandsink x=510 y=0 width=500 height=400 async=true
sync=false

# Run commands to connect to display, and make sure the variables are set
correctly in /etc/default/gstd
# Play the pipeline
gst-client pipeline_play multicam

# Filter message eos
gst-client bus_filter multicam eos

# When done recording, send the EOS event to avoid corruption. Without it,
some header information in the MP4 standard wouldn't be updated and the file
wouldn't be able to be reproduced.
gst-client event_eos multicam

# Wait until bus reads the message eos
gst-client bus_read multicam

# Delete the pipeline when eos received
gst-client pipeline_delete multicam
```

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com or the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.