MediaPipe Reference Manual

Contents

[1 Introduction 4](#_Toc415119865)

[1.1 Terminology 4](#_Toc415119866)

[2 Media Pipe 5](#_Toc415119867)

[2.1 Introduction 5](#_Toc415119868)

[2.2 Get Started 6](#_Toc415119869)

[2.3 Design 7](#_Toc415119870)

[2.4 Types and Values 11](#_Toc415119871)

[2.5 Include 27](#_Toc415119872)

[2.6 Functions 28](#_Toc415119873)

[2.6.1 Pipeline Control 28](#_Toc415119874)

[2.6.2 Channel Control 31](#_Toc415119875)

[2.6.3 Camera Configuration 34](#_Toc415119876)

[2.6.4 Smart Analysis 38](#_Toc415119877)

[2.6.5 Preprocessing 40](#_Toc415119878)

[2.6.6 JPEG Encoding 41](#_Toc415119879)

[2.6.7 H264 Encoding 42](#_Toc415119880)

[2.6.8 OSD 46](#_Toc415119881)

[3 Libjpeg-cl 47](#_Toc415119882)

[3.1 Types and Values 47](#_Toc415119883)

[3.2 Include 49](#_Toc415119884)

[3.3 Functions 50](#_Toc415119885)

Figure

[Figure 1: Media Pipe internal pipeline 7](#_Toc409346254)

# Introduction

This document is helping you to be familiar with Intel’s Media xPipe library for IP camera market including internal implementation and detailed API descriptions.

Media Pipe is designed to speed up the media related application development for IP camera such as camera control and video encoding, etc. Another library named libjpeg-cl is introduced to support JPEG encoding with GPU acceleration.

## Terminology

Table . Terminology

| Term | Description |
| --- | --- |
| API | Application Programming Interface |
| GPU | Graphic Processing Unit |
| OSD | On Screen Display |
| GOP | Group of Picture |
| V4L2 | Video for Linux version 2 |
| CABAC | Context Adaptive Binary Arithmetic Coding |
| JPEG | Joint Photographic Experts Group |
| 3A | Auto Exposure, Auto Focus, Auto White Balance |
| EOS | End of Stream |
| D1 | Digital resolution: 480I 720x480 |
| HD | High Definition Resolution |
| QP | Quantize Parameter |
| VBR | Variable Bit Rate |
| CBR | Constant Bit Rate |
| CPU | Central Processing Unit |

# Media Pipe

## Introduction

Media Pipe is a reusable component for embedded systems that leverages open-source technologies like GStreamer and Open CL to control and accelerate the video capture, pre-processing, encoding, recording and network streaming. Additionally it also supports smart analysis for the captured video such as face detection and recognition.

## Get Started

Media pipe could be run on Intel’s RubyPark IPC SDK image directly. It can be accessed from web browser, below is the steps:

1. Boot into SDK image with root@root
2. Execute below commands:
   1. *cd /etc/mediapipe*
   2. *cp conf.xml conf.xml.bk*
   3. *cp conf.webserver.xml conf.xml*
   4. *cd /etc/ipc-web*
   5. *node bin/www*
3. Open Firefox or chrome to access: ipc\_address:3000，
4. Enter root@root login, then could configure media pipe through web

## Design

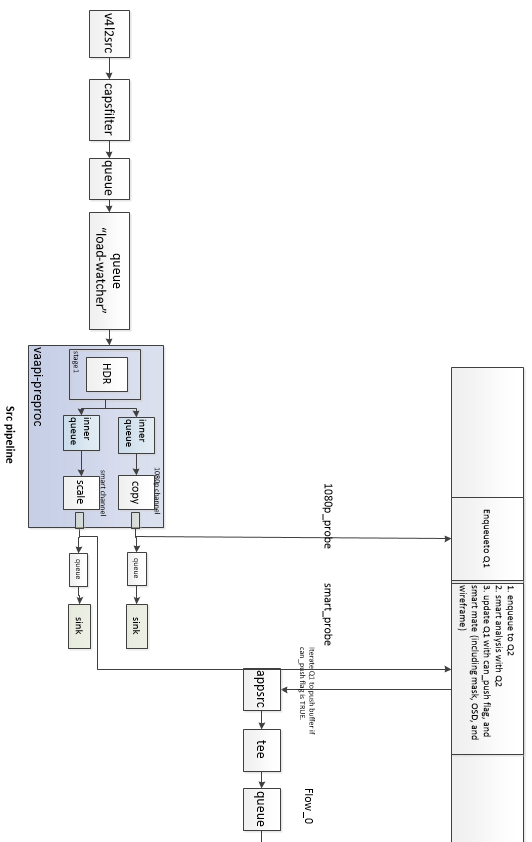
Currently, Media Pipe is designed to have the following features:

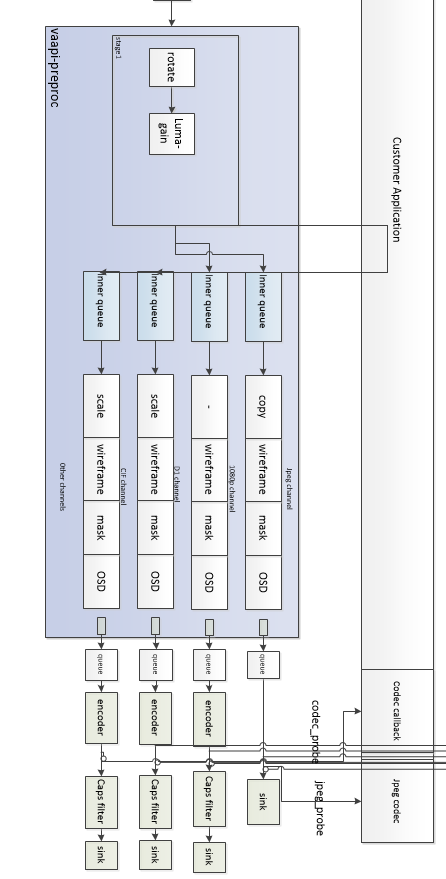
* Video capture through v4l2src
* Video rotation with Open CL acceleration
* Mask, OSD and wireframe adding with Open CL acceleration
* Video recording encoded as H264 with Intel LIBVA acceleration
* Video Snapshot with Open CL based JPEG image compression
* Local video preview

Media Pipe is designed to support multiple video flows, but currently only enable one main flow. One video flow could have multiple video channels with different resolutions or different usages. Except 1080P channel, you’re able to enable or disable any of these channels. And you could also choose where the data of these channels should go. Such as saving as a local file, streaming over network, etc.

The following figure shows the internal Gstreamer pipeline of Media Pipe:

Figure 1: Media Pipe internal pipeline





Media Pipe APIs could be divided into several parts which allows you to control lots of settings from front camera to the last sink for each channel.

By calling below APIs, you could control the Media Pipe itself.

* media\_pipe\_create
* media\_pipe\_destroy
* media\_pipe\_start
* media\_pipe\_stop

To control the camera behavior, you could use the following APIs after choose v4l2src as the source by media\_pipe\_set\_use\_v4l2\_src

* media\_pipe\_set\_v4l2\_src\_sensor\_id
* media\_pipe\_set\_v4l2\_src\_io\_mode
* media\_pipe\_set\_v4l2\_src\_enable\_3a
* media\_pipe\_set\_src\_size
* media\_pipe\_set\_src\_format
* media\_pipe\_set\_src\_frame\_rate
* media\_pipe\_set\_video\_converter\_format

As above figure shows, rotation and luma-gain will be applied to all channels. Other preprocessing including wireframe, mask, OSD can be enabled or disabled for each channel.

Meanwhile you are also able to change the encoding settings such as bit-rate control method, CABAC enabling, the GOP and H264 profile for every channel separately with below APIs:

* media\_pipe\_set\_channel\_encoder\_rate\_control
* media\_pipe\_set\_channel\_encoder\_bitrate
* media\_pipe\_set\_channel\_encoder\_enable\_cabac
* media\_pipe\_set\_channel\_encoder\_gop
* media\_pipe\_set\_channel\_encoder\_profile

## Types and Values

struct \_MediaPipe {

guint size;

gboolean pipe\_running;

};

MediaPipe public context

**Members:**

|  |  |
| --- | --- |
| Size | the size of MediaPipe private context |
| Pipe\_running | MediaPipe running flag |

typedef enum \_SrcType

{

SRC\_TYPE\_VIDEO\_TEST,

SRC\_TYPE\_V4L2,

SRC\_TYPE\_FILE,

SRC\_NUM

}SrcType;

Media Pipe could obtain video from different sources. SRC\_TYPE\_V4L2 is for real IP camera usage case. SRC\_TYPE\_VIDEO\_TEST and SRC\_TYPE\_FILE is for debug.

**Members:**

|  |  |
| --- | --- |
| SRC\_TYPE\_VIDEO\_TEST | Obtain video from Gstreamer videotestsrc element |
| SRC\_TYPE\_V4L2 | Obtain video from camera |
| SRC\_TYPE\_FILE | Obtain video from a H264 ES file |

typedef enum \_MediaSinkType

{

FAKE\_SINK = 0,

FILE\_SINK,

TCP\_SINK,

KMS\_SINK,

V4L2\_SINK,

UDP\_SINK,

SINK\_TYPE\_RESERVE\_1,

SINK\_TYPE\_RESERVE\_2,

SINK\_TYPE\_RESERVE\_3,

SINK\_MAX

}MediaSinkType;

Define how video of each channel is outputted.

**Members:**

|  |  |
| --- | --- |
| FILE\_SINK | Save video into a file. File name is predefined according to the channel internally |
| TCP\_SINK | Stream video over TCP to a specific host and port |
| KMS\_SINK | Output video to a local display |
| FAKE\_SINK | Drop video directly |
| V4L2\_SINK | TBD |
| UDP\_SINK | TBD |

typedef enum \_VideoChannelIndex

{

VIDEO\_CHANNEL\_1080P = 0,

VIDEO\_CHANNEL\_JPEG,

VIDEO\_CHANNEL\_720P,

VIDEO\_CHANNEL\_D1,

VIDEO\_CHANNEL\_CIF,

VIDEO\_CHANNEL\_480\_270,

VIDEO\_CHANNEL\_352\_198,

VIDEO\_CHANNEL\_480\_272,

VIDEO\_CHANNEL\_352\_200,

VIDEO\_CHANNEL\_SMART,

VIDEO\_CHANNEL\_MAX

}VideoChannelIndex;

Media Pipe support multiple channels with different resolution or usage

**Members:**

|  |  |
| --- | --- |
| VIDEO\_CHANNEL\_1080P | Video channel with resolution 1920x1080 |
| VIDEO\_CHANNEL\_JPEG | Video channel with resolution 1920x1080 for JPEG encoding |
| VIDEO\_CHANNEL\_720P | Video channel with resolution 1280x720 |
| VIDEO\_CHANNEL\_D1 | Video channel with resolution 704x576 |
| VIDEO\_CHANNEL\_CIF | Video channel with resolution 352x288 |
| VIDEO\_CHANNEL\_480\_270 | Video channel with resolution 480x270 |
| VIDEO\_CHANNEL\_352\_198 | Video channel with resolution 352x198 |
| VIDEO\_CHANNEL\_480\_272 | Video channel with resolution 480x272 |
| VIDEO\_CHANNEL\_352\_200 | Video channel with resolution 352x200 |
| VIDEO\_CHANNEL\_SMART | Video channel for smart analysis. Resolution is defined by SmartResolution |

typedef enum \_SmartResolution

{

SMART\_RES\_176\_100 = 0,

SMART\_RES\_352\_198,

SMART\_RES\_480\_270,

SMART\_RES\_352\_200,

SMART\_RES\_480\_272,

SMART\_RES\_CIF,

SMART\_RES\_D1

}SmartResolution;

Supported resolution for smart analysis channel. Used by media\_pipe\_set\_src\_smart\_resolution ()

**Members:**

|  |  |
| --- | --- |
| SMART\_RES\_176\_100 | 176x100 as resolution for smart analysis |
| SMART\_RES\_352\_198 | 352x198 as resolution for smart analysis |
| SMART\_RES\_480\_270 | 176x100 as resolution for smart analysis |
| SMART\_RES\_352\_200 | 352x200 as resolution for smart analysis |
| SMART\_RES\_480\_272 | 480x272 as resolution for smart analysis |
| SMART\_RES\_CIF | 352x288 as resolution for smart analysis |
| SMART\_RES\_D1 | 704x576 as resolution for smart analysis |

typedef struct \_SmartData

{

GstBuffer \*buf;

GstVideoPreprocBuffer \*preBuf;

}SmartData;

typedef struct \_Smart1080pData

{

GstBuffer \*buf;

gboolean can\_push;

}Smart1080pData;

Buffer object for smart video frame. In smart analysis callback function

gboolean (\*SmartFrameCallback)(GList \*smart\_queue,

GList \*smart\_1080p\_queue,

gpointer user\_data)

SmartData is the element in smart\_queue, Smart1080pData is the element in smart\_1080p\_queue. Video frame buffer information could be found from SmartData.preBuf such as frame width, height, buffer address etc. can\_push flag in Smart1080pData is used to decide when to push 1080p frame down.

**Members:**

SmartData

|  |  |
| --- | --- |
| Buf | GstBuffer of video frame for smart analysis |
| preBuf | actual used buffer object of smart video frame |

Smart1080pData

|  |  |
| --- | --- |
| Buf | GstBuffer object for 1080P video frame |
| preBuf | flag to control whether to push 1080P video frame down |

typedef struct \_3a\_Config

{

gint flags;

Cameara3a\_WhiteBalance wb;

Cameara3a\_Exposure ep;

Cameara3a\_DeNoise dn;

Cameara3a\_Gamma gm;

Cameara3a\_PicQuality pq;

} Config3A;

Define supported 3A configuration of Media Pipe. Your application needs fill this structure in callback function “Parse3AConfCallback” to tell Media Pipe your preferred 3A configurations.

**Members:**

|  |  |
| --- | --- |
| flags | Refer to Camera3AReconfigFlags, set corresponding flag before you set any detailed configuration to below structure |
| wb | Structure for white balance configurations |
| ep | Structure for exposure configurations |
| dn | Structure for denoise configurations |
| gm | Structure for gamma configurations |
| pg | Structure for picture quality configurations including brightness, contrast, hue, saturation and sharpness |

**Note:**

For all detailed 3A configuration, please refer to libxcam development guide.

typedef enum \_Camera3AReconfigFlags

{

CONFIGFLAG\_3A\_WHITEBALANCE = 0x00000001,

CONFIGFLAG\_3A\_EXPOSURE = 0x00000002,

CONFIGFLAG\_3A\_DENOISE = 0x00000004,

CONFIGFLAG\_3A\_GAMMA = 0x00000008,

CONFIGFLAG\_3A\_PICQUALITY = 0x00000010,

} Camera3AReconfigFlags;

Flags for Config3A.flags.

**Example:**

*// to enable white balance & gamma configuration*

*Config3A config;*

*config.flags |= CONFIGFLAG\_3A\_WHITEBALANCE;*

*config.flags |= CONFIGFLAG\_3A\_GAMMA;*

typedef struct \_Cameara3a\_WhiteBalance

{

gboolean conf\_wb\_mode;

gint val\_wb\_mode;

gboolean conf\_awb\_speed;

gdouble val\_awb\_speed;

gboolean conf\_awb\_cct;

guint val\_awb\_cct\_min;

guint val\_awb\_cct\_max;

gboolean conf\_awb\_gm;

gdouble val\_awb\_gr;

gdouble val\_awb\_r;

gdouble val\_awb\_gb;

gdouble val\_awb\_b;

} Cameara3a\_WhiteBalance;

White balance configuration.

**Members:**

|  |  |
| --- | --- |
| conf\_wb\_mode | Enable white balance mode configuration or not. TRUE to enable |
| val\_wb\_mode | White balance mode:   |  |  | | --- | --- | | -1 | Not set | | 0 | Auto mode | | 1 | Manual mode | | 2 | Daylight mode | | 3 | Sunset mode | | 4 | Cloudy mode | | 5 | Tungsten mode | | 6 | Fluorescent mode | | 7 | Warm fluorescent mode | | 8 | Shadow mode | | 9 | Warm incandescent mode | |
| conf\_awb\_speed | Enable auto white balance speed configuration or not. TRUE to enable |
| val\_awb\_speed | Auto white balance speed. 0.0~1.0 |
| conf\_awb\_cct | Enable color temperature configuration or not. Valid in auto white balance mode  Range: 0 ~ 10000  Default: min: 0; max: 0 |
| val\_awb\_cct\_min | Minimum color temperature. |
| val\_awb\_cct\_max | Maximum color temperature. |
| conf\_awb\_gm | Enable manual color gain configuration or not. |
| val\_awb\_gr | GR channel gain  Range: 0.1 ~ 4.0  0 to disable |
| val\_awb\_r | Red channel gain  Range: 0.1 ~ 4.0  0 to disable |
| val\_awb\_gb | GB channel gain  Range: 0.1 ~ 4.0  0 to disable |
| val\_awb\_b | Blue channel gain  Range: 0.1 ~ 4.0  0 to disable |

typedef struct \_Cameara3a\_Exposure

{

gboolean conf\_ep\_mode;

gint val\_ep\_mode;

gboolean conf\_meter\_mode;

gint val\_meter\_mode;

gboolean conf\_ep\_window;

XCam3AWindow val\_ep\_window;

gboolean conf\_ep\_offset;

gdouble val\_ep\_offset;

gboolean conf\_ep\_flicker;

gint val\_ep\_flicker;

gboolean conf\_ep\_manual\_time;

gint64 val\_ep\_manual\_time;

gboolean conf\_ep\_manual\_analoggain;

gdouble val\_ep\_manual\_analoggain;

gboolean conf\_ep\_max\_analoggain;

gdouble val\_ep\_max\_analoggain;

gboolean conf\_ep\_timerange;

gint64 val\_ep\_timerange\_min;

gint64 val\_ep\_timerange\_max;

}Cameara3a\_Exposure;

**Members:**

|  |  |
| --- | --- |
| conf\_ep\_mode | Exposure mode configuration flag. True to enable |
| val\_ep\_mode | |  |  | | --- | --- | | 0 | Auto mode | | 1 | Manual expose | | 2 | Shutter priority expose | | 3 | Aperture priority expose | |
| conf\_meter\_mode | Metering mode configuration flag. True to enable. |
| val\_meter\_mode | |  |  | | --- | --- | | 0 | Auto mode | | 1 | Spot metering mode | | 2 | Center metering mode | |
| conf\_ep\_window | Expose window configuration flag. True to set. |
| val\_ep\_window | The area of the expose window. Only valid in spot metering mode. |
| conf\_ep\_offset | Expose offset configuration flag. True to enable |
| val\_ep\_offset | Expose offset value   * Range: -4.0 ~ 4.0. * Default: 0.0 |
| conf\_ep\_flicker | Expose flicker mode configuration flag. True to enable |
| val\_ep\_flicker | |  |  | | --- | --- | | 0 | Auto flicker mode | | 1 | 50Hz mode | | 2 | 60Hz mode | | 3 | Flicker mode off | |
| conf\_ep\_manual\_time | Manual expose time configuration |
| val\_ep\_manual\_time | In microseconds.  Range: 0 ~ 1/fps |
| conf\_ep\_manual\_analoggain | Manual analog gain configuration flag |
| val\_ep\_manual\_analoggain | Analog gain value.  Range: 1.0 ~ 4.0  Default: 0.0 to disable |
| conf\_ep\_max\_analoggain | Max manual analog gain configuration flag |
| val\_ep\_max\_analoggain | Max manual analog gain value  Range: 1.0 ~ 4.0  Default: 0.0 to disable |
| conf\_ep\_timerange | Auto expose time range configuration flag |
| val\_ep\_timerange\_min | Minimum auto expose time in microseconds  Range: 0 ~ 1000000/fps  Default: 0 to disable |
| val\_ep\_timerange\_max | Maximum auto expose time in microseconds  Range: 0 ~ 1000000/fps  Default: 0 to disable |

typedef struct \_Cameara3a\_DeNoise

{

gboolean conf\_dn\_level;

gint val\_dn\_level;

}Cameara3a\_DeNoise;

De noise configuration.

**Members:**

|  |  |
| --- | --- |
| conf\_dn\_level | Noise reduction configuration flag. True to enable |
| val\_dn\_level | Noise reduction level for BNR and YNR.  Range: 0~255. Default: 128 |

typedef struct \_Cameara3a\_Gamma

{

gboolean conf\_gm\_table;

double val\_gm\_table\_r[GAMMATABLESIZE];

double val\_gm\_table\_g[GAMMATABLESIZE];

double val\_gm\_table\_b[GAMMATABLESIZE];

gboolean conf\_gm\_gbce;

gboolean val\_gm\_gbce;

}Cameara3a\_Gamma;

Gamma configuration.

**Members:**

|  |  |
| --- | --- |
| conf\_gm\_table | Gamma configuration flag. True to enable |
| val\_gm\_table\_r | Gamma table for red channel.  Range: 0.0 ~ 1.0, GAMMATABLESIZE is 256. |
| val\_gm\_table\_g | Gamma table for green channel |
| val\_gm\_table\_b | Gamma table for blue channel |
| conf\_gm\_gbce | GBCE (Global Brightness Contrast Enhancement) configuration flag. True to enable |
| val\_gm\_gbce | 1 to enable |

typedef struct \_Cameara3a\_PicQuality

{

gboolean conf\_pq\_brightness;

guint8 val\_pq\_brightness;

gboolean conf\_pq\_contrast;

guint8 val\_pq\_contrast;

gboolean conf\_pq\_hue;

guint8 val\_pq\_hue;

gboolean conf\_pq\_saturation;

guint8 val\_pq\_saturation;

gboolean conf\_pq\_sharpness;

guint8 val\_pq\_sharpness;

}Cameara3a\_PicQuality;

Picture quality configuration.

**Members:**

|  |  |
| --- | --- |
| conf\_pq\_brightness | Brightness configuration flag. True to enable |
| val\_pq\_brightness | Brightness value.   * Range: 0~255. * Default: 128 |
| conf\_pq\_contrast | Contrast configuration flag. True to enable |
| val\_pg\_contrast | Contrast value.   * Range: 0~255. * Default: 128 |
| conf\_pq\_hue | Hue configuration flag. True to enable |
| val\_pq\_hue | Hue value.   * Range: 0~255. * Default: 128 |
| conf\_pq\_saturation | Saturation configuration flag. True to enable |
| val\_pq\_saturation | Saturation value.   * Range: 0~255. * Default: 128 |
| conf\_pq\_sharpness | Sharpness configuration flag. True to enable |
| val\_pq\_sharpness | Sharpness value.   * Range: 0~255. * Default: 128 |

typedef gboolean (\*Parse3AConfCallback) (Config3A \*ptrConf, gpointer user\_data);

3A configuration callback. Fill Config3A structure to set or change all 3A configurations in this callback.

typedef gboolean (\*EncodeFrameCallback) (GstBuffer \*encode\_buf, gpointer user\_data, VideoChannelIndex channel);

Specified by media\_pipe\_set\_channel\_encoder\_frame\_callback. You could get H264 encoded buffer of one channel in this callback, then do change or analysis.

**Members:**

|  |  |
| --- | --- |
| encode\_buf | H264 encoded buffer as GstBuffer format. |
| user\_data | User specified pointer |
| channel | Channel index |

typedef gboolean (\*VideoFrameCallback) (GstVideoPreprocBuffer \*video\_buf, gpointer user\_data, VideoChannelIndex channel);

You could get video frame buffer of one channel in this callback.

typedef gboolean (\*SmartFrameCallback) (GList \*smart\_queue, GList \*smart\_1080p\_queue, gpointer user\_data);

Specified by media\_pipe\_set\_src\_frame\_smart\_callback. You could get video data in YUV format from smart video channel to do smart analysis. Refer to media\_pipe\_set\_src\_frame\_smart\_callback for more details.

typedef gboolean (\*MessageCallback) (GstMessage \*mesg, gpointer user\_data);

Message callback function. Specified by media\_pipe\_set\_message\_callback. It will be called when EOS, error or other things occur in the internal pipeline.

## Include

#include <mediapipe.h>

## Functions

### Pipeline Control

MediaPipe

\*media\_pipe\_create (int argc, char \*agrv[]);

Initialize the MeidaPipe and Gstreamer library. All Gstreamer commandline options are supported.

**Parameters:**

|  |  |
| --- | --- |
| argc | application's argc |
| argv | application's argv |

**Returns:**

A pointer to MediaPipe context, assert if memory couldn't be allocated.

void

media\_pipe\_destroy (MediaPipe \*pipe);

Clean up any resources created by media\_pipe\_create, destroy internal Gstreamer pipeline.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MeidaPipe context allocated by media\_pipe\_create |

gboolean

media\_pipe\_start (MediaPipe \*pipe);

Build and start the internal Gstreamer pipeline, run a Glib main loop until pipeline occurs errors or receives EOS.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |

**Returns:**

FALSE indicates errors

**Note:**

Can't start again after stop

Void

media\_pipe\_stop (MediaPipe \*pipe);

Stop the Glib main loop by sending an EOS to internal Gstreamer pipeline.

Any calls to media\_pipe\_start will return.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |

void media\_pipe\_set\_message\_callback (MediaPipe \*pipe,

MessageCallback callback,

gpointer user\_data);

Specify callback to receive message from internal main pipeline bus including EOS, errors etc.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| user\_data | User specified pointer |

gboolean

media\_pipe\_set\_filesrc\_type (MediaPipe \*pipe, SrcType src\_type);

Choose video source.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| src\_type | Three sources could be selected   |  |  | | --- | --- | | SRC\_TYPE\_VIDEO\_TEST | Video from 'videotestsrc' element, for debug only when there is no camera available. | | SRC\_TYPE\_V4L2 | Video from camera, normal usage | | SRC\_TYPE\_FILE | Video from H264 raw file, for debug | |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

gboolean

media\_pipe\_set\_filesrc\_location (MediaPipe \*pipe, gchar \*location);

Specify the file location when choose SRC\_TYPE\_FILE as video source

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| location | H264 raw file path |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

Gboolean

media\_pipe\_set\_use\_v4l2\_src (MediaPipe \*pipe, gboolean use\_v4l2\_src);

Enable v4l2src element or not. It's disabled by default. If you need to capture data from camera,

Set this before media\_pipe\_start. Otherwise MediaPipe will take videotestsrc as source for debugging.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| use\_v4l2\_src | Enable v4l2src element or not |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

### Channel Control

Gboolean

media\_pipe\_enable\_video\_channel(

MediaPipe \*pipe,

VideoChannelIndex channel);

Enable one channel. Need to be called before media\_pipe\_start. The 1080P channel is enabled by default.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |

**Returns:**

TRUE if setting is successfully. FALSE if not.

Gboolean

media\_pipe\_disable\_video\_channel(

MediaPipe \*pipe,

VideoChannelIndex channel);

Disable one channel. The 1080P channel could not be disabled.

Need to be called before media\_pipe\_start too.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |

**Returns:**

TRUE if setting is successfully. FALSE if not.

Gboolean media\_pipe\_set\_channel\_sink\_type(

MediaPipe \*pipe,

VideoChannelIndex channel,

MediaSinkType sink\_type,

gchar \*host\_ip,

gint port);

Choose sink type for a channel. It determines where video data (YUV or encoded data depends on the sink type) goes. Currently support list:

* FAKE\_SINK
* FILE\_SINK
* TCP\_SINK
* KMS\_SINK
* RTSP\_SINK

Type FAKE\_SINK will simply drop the video data.

Type FILE\_SINK will save the video data of this channel as a file in the same directory as the application using MediePipe. The file name for each channel is shown in the following table:

|  |  |
| --- | --- |
| VIDEO\_CHANNEL\_1080P | 1080p.264 |
| VIDEO\_CHANNEL\_JPEG | jpeg |
| VIDEO\_CHANNEL\_720P | 720p.264 |
| VIDEO\_CHANNEL\_D1 | d1.264 |
| VIDEO\_CHANNEL\_CIF | cif.h264 |
| VIDEO\_CHANNEL\_480\_270 | 480x270.264 |
| VIDEO\_CHANNEL\_352\_198 | 352x198.264 |
| VIDEO\_CHANNEL\_480\_272 | 480x272.264 |
| VIDEO\_CHANNEL\_352\_200 | 352x200.264 |
| VIDEO\_CHANNEL\_SMART | smart.264 |

TCP\_SINK will stream video data (YUV or encoded video depends on the configuration of this channel) of one channel to a specified host which provided by host\_ip and port.

KMS\_SINK is for local preview. Need to disable the encoding of this channel when choosing this type.

RTSP\_SINK will setup a RTSP server. Encoding is automatically enabled when choosing this type. The RTSP server addresses for each channel are different. Refer to the following table:

|  |  |
| --- | --- |
| VIDEO\_CHANNEL\_1080P | rtsp://ipc\_address:8554/1080p |
| VIDEO\_CHANNEL\_JPEG | Not support |
| VIDEO\_CHANNEL\_720P | rtsp://ipc\_address:8554/720p |
| VIDEO\_CHANNEL\_D1 | rtsp://ipc\_address:8554/d1 |
| VIDEO\_CHANNEL\_CIF | rtsp://ipc\_address:8554/cif |
| VIDEO\_CHANNEL\_480\_270 | rtsp://ipc\_address:8554/480\_270 |
| VIDEO\_CHANNEL\_352\_198 | rtsp://ipc\_address:8554/352\_198 |
| VIDEO\_CHANNEL\_480\_272 | rtsp://ipc\_address:8554/480\_272 |
| VIDEO\_CHANNEL\_352\_200 | rtsp://ipc\_address:8554/352\_200 |
| VIDEO\_CHANNEL\_SMART | rtsp://ipc\_address:8554/smart |

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |
| sink\_type | Sink type |
| host\_ip | Specify host IP address when chooses TCP\_SINK |
| port | Specify host port when chooses TCP\_SINK |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

Gboolean media\_pipe\_set\_channel\_encoder\_frame\_callback (

MediaPipe \*pipe,

VideoChannelIndex channel,

EncodeFrameCallback callback,

gpointer user\_data);

Set callback to get encoded frame of a channel.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |
| callback | Callback function |
| user\_data | User data to be passed to the callback function |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

### Camera Configuration

Gboolean media\_pipe\_set\_v4l2\_src\_sensor\_id (

MediaPipe \*pipe,

guint v4l2\_src\_sensor\_id);

Choose the type of sensors. Get more details by running 'gst-inspect-1.0 v4l2src' on target, check 'sensor-id' property.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| v4l2\_src\_sensor\_id | Sensor id   |  |  | | --- | --- | | 0 | ov5640\_1 | | 1 | aptina | | 2 | ov5640\_2 | | 3 | imx185 | |

**Returns:**

FALSE if MediaPipe is initialized or already running, otherwise TRUE.

Gboolean media\_pipe\_set\_v4l2\_src\_io\_mode (

MediaPipe \*pipe,

guint v4l2\_src\_io\_mode);

Set IO mode for v4l2src element. Generally v4l2src supports four types of IO mode: RW, MMAP, USERPTR, DMABUF.

But not every v4l2 driver supports all of them. For better performance, MediaPipe chooses DMABUF by default for all supported devices.

**Parameters:**

|  |  |  |
| --- | --- | --- |
| pipe | MediaPipe context | |
| v4l2\_src\_io\_mode | | V4l2src IO mode   |  |  | | --- | --- | | 0 | Auto | | 1 | RW (Read and Write) | | 2 | Memory Map Mode | | 3 | Userptr Mode | | 4 | DMABUF mode | | |

**Returns:**

FALSE if MediaPipe is initialized or already running, otherwise TRUE.

Gboolean media\_pipe\_set\_v4l2\_src\_capture\_mode (

MediaPipe \*pipe,

guint capture\_mode);

Choose capture mode for v4l2src element. Currently only support video mode.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| capture\_mode | |  |  | | --- | --- | | 0 | Still mode | | 1 | Video mode | | 2 | Preview mode | |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

Gboolean media\_pipe\_set\_src\_size ( MediaPipe \*pipe,

guint width,

guint height);

Set resolution for the source element which could be v4l2src or videotestsrc. Default setting is 1920x1080. Unsupported resolution won't cause errors in this function immediately. It will be reported in media\_pipe\_start.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| width | Source width in pixel |
| height | Source height in pixel |

**Returns:**

FALSE if MediaPipe is initialized or already running, otherwise TRUE.

Gboolean

media\_pipe\_set\_src\_frame\_rate(MediaPipe \*pipe, guint frame\_rate);

Set global frame rate for MediaPipe. It's 30fps by default.

Example: set frame\_rate to 30 to get 30fps.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| frame\_rate | Numerator of the frame rate, denominator is always 1 |

**Returns:**

FALSE if MediaPipe is running, otherwise TRUE.

gboolean

media\_pipe\_set\_src\_format ( MediaPipe \*pipe,

GstVideoFormat format);

Set color format for source element. Refer to GstVideoFormat manual for details. You can't choose any format as you like, it largely depends on the sensor and its driver. Default setting is NV12.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| format | video color format, refer to GstVideoFormat from Gstreamer programming manual |

**Returns:**

FALSE if MediaPipe is initialized or already running, otherwise TRUE.

gboolean

media\_pipe\_set\_v4l2\_src\_enable\_3a (MediaPipe \*pipe, gboolean enable\_3a);

Enable 3A for v4l2src element, disabled by default. Don't support eanble after MediaPipe is already running.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| enable\_3a | True to enable |

**Returns:**

FALSE indicates errors.

gboolean

media\_pipe\_set\_src\_parse\_3aconf\_callback (

MediaPipe \*pipe,

Parse3AConfCallback parse\_3aconf\_callback,

gpointer user\_data);

Set callback function for 3A configuration. In the callback, you should fill Config3A structure with all or part of 3A configuration including white balance, exposure, denoise, gamma, picture quality settings.

This callback will be called in media\_pipe\_reconfig\_3a.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| parse\_3aconf\_callback | Callback function pointer |
| user\_data | Customized data pointer passed to the callback |

**Returns:**

FALSE if MediaPipe is already running or other errors, otherwise TRUE.

gboolean

media\_pipe\_reconfig\_3a (MediaPipe \*pipe);

Reconfigure 3A settings. It will call callback function set by media\_pipe\_set\_parse\_3aconf\_callback to get new settings, then do reconfiguration. It can only be called when MediaPipe is running and 3A is enabled by media\_pipe\_set\_v4l2\_src\_enable\_3a.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |

**Returns:**

FALSE indicates errors. Normally 3A is disabled, Parse3AConfCallback returns errors would lead to errors.

### Smart Analysis

Mediapipe allows users to get the video data from the specified callback function (set by media\_pipe\_set\_src\_frame\_smart\_callback) to do analysis such as face detection, face recognition and other kinds of analysis. It’s designed to provide a separate channel independent from other streaming channels for smart analysis. Users can choose the video resolution of smart analysis channel by media\_pipe\_set\_src\_smart\_resolution.

gboolean

media\_pipe\_set\_src\_smart\_resolution (

MediaPipe \*pipe,

SmartResolution resolution);

Choose a resolution to do the smart video analytic. For performance issue, we only support analytic for smaller resolution below D1 instead of the full HD video.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| resolution | Resolution for smart analysis. Refer to SmartResolution |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

gboolean

media\_pipe\_set\_src\_frame\_smart\_callback (

MediaPipe \*pipe,

SmartFrameCallback smart\_analyze\_callback,

gpointer user\_data);

Set callback function for smart analysis.

gboolean (\*SmartFrameCallback)(GList \*smart\_queue,

GList \*smart\_1080p\_queue,

gpointer user\_data)

You can get video frames from smart\_queue for analysis. Every buffer is stored as a SmartData.

Detailed buffer information could be found from SmartData.preBuf such as frame width, height, buffer address etc.

smart\_1080p\_queue is a queue of Smart1080pData, you could design customized policy to push 1080p frame down by setting Smart1080pData->can\_push as TRUE.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| smart\_analyze\_callback | Callback function pointer |
| user\_data | User customized data pointer |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

### Preprocessing

gboolean

media\_pipe\_set\_video\_preproc\_rotation (MediaPipe \*pipe,

GstVideoPreprocRotateMode rotation\_mode);

Set rotation mode all channels.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| rotation\_mode | Refer to GstVideoPreprocRotateMode |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

gboolean

media\_pipe\_set\_video\_preproc\_luma (MediaPipe \*pipe,

guint gain);

Set luminance for all channels.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| gain | Percentage of original luminance. 100 means 100% |

**Returns:**

FALSE indicates errors.

### JPEG Encoding

gboolean

media\_pipe\_set\_video\_preproc\_frame\_jpeg\_callback (

MediaPipe \*pipe,

VideoFrameCallback jpeg\_callback,

gpointer user\_data);

Set callback functions to do the JPEG encoding.

gboolean (\*VideoFrameCallback)(GstVideoPreprocBuffer \*video\_buf,

gpointer user\_data,

VideoChannelIndex channel);

* video\_buf contains information required for JPEG encoding such as frame resolution, buffer address etc.
* user\_data user's own data
* channel always VIDEO\_CHANNEL\_1080P

Video frame here is copy from 1080P channel. Normally, try libjpeg-cl to do the encoding with hardware acceleration.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| jpeg\_callback | Callback function pointer |
| User\_data | User’s data pointer |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

### H264 Encoding

gboolean

media\_pipe\_set\_channel\_encoder\_toggle(

MediaPipe \*pipe,

VideoChannelIndex channel,

gboolean enable);

Enable or disable H264 encoding for one channel. It's enabled by default.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |
| enable | Enable flag |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

gboolean

media\_pipe\_set\_channel\_encoder\_rate\_control (

MediaPipe \*pipe,

VideoChannelIndex channel,

guint rate\_control);

Choose the rate control method for internal H264 encoder.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |
| rate\_control | Rate control method, discrete value can be chosen   |  |  | | --- | --- | | 1 | Constant QP | | 2 | CBR | | 4 | VBR | | 5 | VBR constrained | |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

gboolean

media\_pipe\_set\_channel\_encoder\_bitrate (

MediaPipe \*pipe,

VideoChannelIndex channel,

guint bitrate);

Set bit rate of internal H264 encoder for one channel. This could be set dynamically while Media Pipe is running.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |
| bitrate | Bitrate in kbps. Range: 0~102400 |

**Returns:**

FALSE indicates errors

gboolean

media\_pipe\_set\_channel\_encoder\_enable\_cabac(

MediaPipe \*pipe,

VideoChannelIndex channel,

gboolean enable);

Enable or disable CABAC for internal H264 encoder. It's disabled by default.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |
| enable | Enable flag |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

gboolean

media\_pipe\_set\_channel\_encoder\_mv (

MediaPipe \*pipe,

VideoChannelIndex channel,

guint mv);

Enable or disable motion vector dump for a specific video channel. Except the 1080P channel, motion vector for other channels is disabled by default. There are 2 types of motion vectors to enable: 16x16 block, 4x4 block.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |
| mv | Mode   |  |  | | --- | --- | | 0 | disable | | 1 | enable with 16x16 mv block | | 2 | enable with 4x4 mv block | |

**Returns:**

FALSE if MediaPipe already running, otherwise TRUE.

gboolean

media\_pipe\_set\_channel\_encoder\_gop(

MediaPipe \*pipe,

VideoChannelIndex channel,

guint m,

guint n);

Set key frames period and maximal B-frames between I and P of H264 encoder for one channel.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |
| m | Key frames period   * Range: 0-300 * Default: 0 for auto-calculate |
| n | Maximal B-frames between I and P   * Range: 0-10 * Default: 0 |

**Returns:**

FALSE if MediaPipe already running, otherwise TRUE

gboolean

media\_pipe\_set\_channel\_encoder\_gop(

MediaPipe \*pipe,

VideoChannelIndex channel,

guint m,

guint n);

Choose H264 encoder profile for one channel.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |
| Profile | H264 profile   |  |  | | --- | --- | | 0 | baseline(default) | | 1 | main | | 2 | high | |

**Returns:**

FALSE if MediaPipe already running, otherwise TRUE

gboolean

media\_pipe\_set\_channel\_key\_frame(MediaPipe \*pipe,

VideoChannelIndex channel);

Trigger a key frame event for a specific channel. Must be called when MediaPipe is running and specified channel is on.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Channel number |

**Returns:**

FALSE if channel is off or MediaPipe isn't running, otherwise TRUE

### OSD

gboolean

media\_pipe\_set\_channel\_enable\_osd (

MediaPipe \*pipe,

VideoChannelIndex channel,

gboolean enable);

Enable OSD for one channel.

**Parameters:**

|  |  |
| --- | --- |
| pipe | MediaPipe context |
| channel | Video channel number |
| enable | True to enable |

**Returns:**

FALSE if MediaPipe is already running, otherwise TRUE.

# Libjpeg-cl

## Types and Values

typedef void\* sapp\_context;

Context of lib3app

typedef enum {

USE\_OPENCL = 0,

USE\_LIBJPEG = 1

}JPEG\_TYPE;

Method of jpeg encoding

**Members:**

|  |  |
| --- | --- |
| USE\_OPENCL | Using OpenCL to encode jpeg. GPU accelerated |
| USE\_LIBJPEG | Using libjpeg to encode jpeg. CPU rendering |

typedef struct {

void\* handle;

unsigned int w;

unsigned int h;

unsigned int ystride;

unsigned int uvstride;

unsigned int yoffset;

unsigned int uvoffset;

}JpegInputBuffer;

Structure of input buffer for jpeg encoding.

**Members:**

|  |  |
| --- | --- |
| handle | Handle of NV12 buffer. It is a pointer of unmapped drm\_intel\_bo |
| w | Width of input buffer (in pixel) |
| h | Height of input buffer (in pixel) |
| ystride | Stride of Y planar (in byte) |
| uvstride | Stride of UV planar (in byte) |
| yoffset | Offset of Y planar (in byte) |
| uvoffset | Offset of UV planar (in byte) |

typedef struct {

unsigned int x;

unsigned int y;

unsigned int w;

unsigned int h;

}JpegCropRect;

Structure of region which shall be encoded.

**Members:**

|  |  |
| --- | --- |
| x | Start x of encoded region (in pixel) |
| y | Start y of encoded region (in pixel) |
| w | Width of encoded region (in pixel) |
| h | Height of encoded region (in pixel) |

typedef struct {

unsigned char\* output\_buffer;

size\_t output\_size;

size\_t header\_size;

}JpegOutputBuffer;

Structure of output buffer for jpeg encoding, where jpeg data is stored.

**Members:**

|  |  |
| --- | --- |
| output\_buffer | Pointer of jpeg compressed data buffer |
| output\_size | Size of jpeg compressed data buffer |
| header\_size | Size of jpeg header |

## Include

#include “jpeglib\_interface.h”

## Functions

int jpeg\_init(JPEG\_TYPE type);

Initialise and indicate jpeg encoding type.

**Parameters:**

|  |  |
| --- | --- |
| type | Type of jpeg encoder   * USE\_OPENCL * USE\_LIBJPEG |

**Returns:**

0 in any case.

int jpeg\_encode\_file(JpegInputBuffer\* src,

int quality,

const char\* dst,

char\* exif,

size\_t exif\_size,

JpegCropRect\* rect);

Encode jpeg into a picture file.

**Parameters:**

|  |  |
| --- | --- |
| src | Input buffer where raw nv12 data is stored |
| quality | Control jpeg quality which takes effect in quantization scaling |
| dst | Name of jpeg file |
| exif | Pointer of exif buffer (disabled internally) |
| exif\_size | Size of exif buffer (disabled internally) |
| rect | Region which will be encoded |

**Returns:**

0 in any case.

int jpeg\_encode\_buffer(JpegInputBuffer\* src,

int quality,

char\* exif,

size\_t exif\_size,

JpegOutputBuffer\* dst);

Encode the whole buffer into jpeg output buffer. Note: Only available when you choose USE\_OPENCL.

**Parameters:**

|  |  |
| --- | --- |
| src | Input buffer where raw nv12 data is stored |
| quality | Control jpeg quality which takes effect in quantization scaling |
| exif | Pointer of exif buffer (disabled internally) |
| exif\_size | Size of exif buffer (disabled internally) |
| dst | Output buffer |

**Returns:**

0 in any case.

int jpeg\_encode\_crop\_buffer(JpegInputBuffer\* src,

int quality,

char\* exif,

size\_t exif\_size,

JpegOutputBuffer\* dst,

JpegCropRect\* cropRect);

Encode a region into jpeg output buffer. Note: Only availble when you choose USE\_OPENCL.

**Paramters:**

|  |  |
| --- | --- |
| src | Input buffer where raw nv12 data is stored |
| quality | Control jpeg quality which takes effect in quantization scaling |
| exif | Pointer of exif buffer (disabled internally) |
| exif\_size | Size of exif buffer (disabled internally) |
| dest | Output buffer |
| rect | Region which will be encoded |

**Returns:**

0 in any case.

Destroy output buffer. Used only when decoding jpeg into an output buffer. Never use it when decoding into picture file.

int jpeg\_destroy\_buffer();

**Returns:**

0 in any case.

int jpeg\_release();

Release and cleanup occupied resources.

**Returns:**

0 in any case.