MXUVC Programming Interface for IP Camera Application

August 29, 2013
Document Revision: 0.5

GEO Semiconductor Inc.
2350 Mission College Blvd., Suite 1050
Santa Clara, CA 94054
United States
Ph: 408-844-8800

GEO Semiconductor Inc.
155 Gordon Baker Rd., Suite 201
Toronto, Ontario M2J 5B4
Canada
Ph: 647-260-1232

www.geosemi.com
infogeo@geosemi.com

For pricing, delivery, and ordering information, please contact GEO Semiconductor Direct at 408-844-8800 or visit GEO's website at www.geosemi.com.

1. INTRODUCTION

MXUVC is a simple, user-friendly application interface to control and capture audio and multi-channel video from a **Geo Camera** over the USB on a host platform**.**

As shown in the **Figure 1.1** below, **Geo Cameras** are connected to the Host system through the USB. **Geo Cameras** are USB Video Class (**UVC**) and USB Audio Class (**UAC**) compatible. The Video and Audio functionalities of the Geo **Cameras** can be accessed through the standard Linux V4L2 UVC Video Interface and the ALSA UAC Audio Interface.

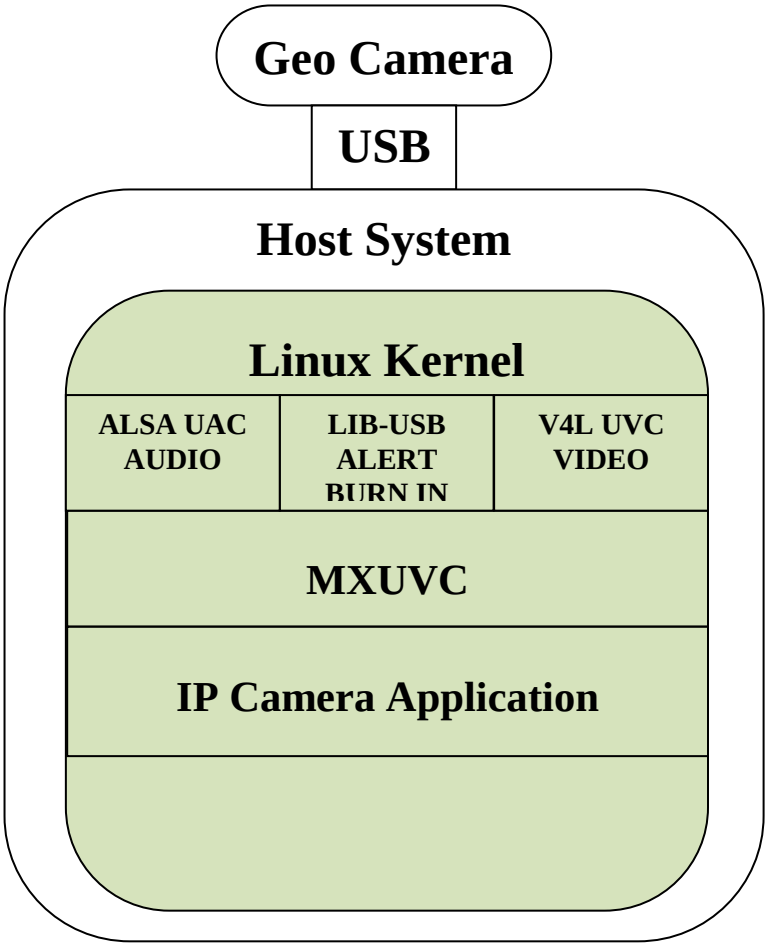

Figure 1.1 *Geo Camera-host system interfaces for IP camera Application.*

MXUVC can be broadly divided into 3 subsystems namely **Video Subsystem, Audio Subsystem and Alert Subsystem.** MXUVC provides an IP Camera application framework over the standard Linux V4l2 UVC Video and the ALSA UAC Audio interface forming the Video and the Audio subsytems. The Alert subsystem is a customized interface over the linux lib-usb layer implemented using vendor specific commands and a dedicated usb end-point. The lib-usb interface is also used to overlay customary text in the video **(Burn-In)**. One use-case for the **Burn-In** text feature would be to over-lay the capture time in the video.

MXUVC at high level provides the below listed functionalities by which the user can develop IP Camera application on the host system using a **Geo camera.**

1. Control and Capture Multi-Channel Video through Video Subsystem.
2. Tuning the Sensor and Video parameters through the Video Subsystem.
3. Control and Capture Audio through the Audio Subsystem.
4. Configuring and receiving motion and audio alert parameters through the Alert sub system.
5. Overlaying customary text in the video, to enable features like overlaying capture-time etc for predefined font.

MXUVC APIs for these sub systems are discussed in great detail in the subsequent chapters.

2. MXUVC VIDEO SUBSYSTEM

MXUVC Video subsystem is used to initialize the Geo camera Video Interface, configuring the video and sensor parameters and start capturing the video data for further processing. MXUVC supports capture of multiple channels of video data of varying resolution and format. However the resolution and format supported by a channel is fixed for a product configuration. The Data Structures and the APIs are explained in the below sections

2.1. Data Structures

2.1.1. *video_channel_t*

Description:

Enumeration to indicate the video channel number.

Declaration:

```
typedef enum channel {
      ...
      /* channels for ip camera */
      CH1  = 0,
      CH2  = 1,
      CH3  = 2,
      CH4  = 3,
      CH5  = 4,
      CH6  = 5,
      CH7  = 6,
      NUM_MUX_VID_CHANNELS  = CH7+1,
      CH_RAW            = NUM_MUX_VID_CHANNELS,
      NUM_IPCAM_VID_CHANNELS
} video_channel_t;
```

2.1.2. *video_profile_t*

Description:

Enumeration to indicate the H264 profile used for encoding.

Declaration:

```
typedef enum {
      PROFILE_BASELINE = 0,
      PROFILE_MAIN     = 1,
      PROFILE_HIGH     = 2,
      NUM_PROFILE
} video_profile_t;
```

2.1.3. *video_format_t*

Description:

Enumeration of video format used in a particular channel

Declaration:

```
typedef enum {
      FIRST_VID_FORMAT       = 0,
      VID_FORMAT_H264_RAW    = 0,
```

```
        VID_FORMAT_H264_TS      = 1,
        VID_FORMAT_MJPEG_RAW    = 2,
        VID_FORMAT_YUY2_RAW     = 3,
        VID_FORMAT_NV12_RAW     = 4,
        VID_FORMAT_GREY_RAW     = 5,
        VID_FORMAT_H264_AAC_TS  = 6,
        VID_FORMAT_MUX          = 7,
        NUM_VID_FORMAT
} video_format_t;
```

2.1.4. *video_channel_info_t*

Description:

Structure containing the information regarding an encoding channel.

Declaration:

```
typedef struct {
        video_format_t      format;
        uint16_t            width;
        uint16_t            height;
        uint32_t            framerate;
        uint32_t            goplen;
        video_profile_t     profile;
        uint32_t            bitrate;
        uint32_t            compression_quality;
}video_channel_info_t;
```

Variables:

format	Format of the video used in the channel
width	Width of the video in the channel.
height	Height of the video in the channel.
framerate	Frame Rate of the video in the channel.
goplen	GOP size of video to be used in the channel, applicable only for VID_FORMAT_H264_RAW, *VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.*
profile	H264 profile used for encoding in the channel. Applicable only for VID_FORMAT_H264_RAW, VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.
bitrate	Bitrate of the video in the channel.
compression_quality	Compression quality in terms of the QP factor set for the video on this channel, applicable only for VID_FORMAT_MJPEG_RAW.

2.1.5. *motion_stat_t*

Description:

Structure containing the motion vector statistics of the video.

Declaration:

```
typedef struct {
        uint8_t*    buf;
        int         size;
}motion_stat_t;
```

Variables:

buf	pointer to the motion vector statistics buffer.
size	size of the buffer.

2.1.6. *video_info_t*

Description:

Video Information structure used for processing the video data received from the camera in the call back function.

Declaration:

```
typedef struct {
      video_format_t     format;
      uint64_t           ts;
      motion_stat_t      stats;
      int                buf_index;
} video_info_t;
```

Variables:

format	Format of the video frame received.
ts	Video frame timestamp in terms of ticks of 90kHz clock, where each tick corresponds to 1/(90 * 1000) sec or 1/90 ms.
stats	motion vector statistics information of the video frame. Present only in case of VID_FORMAT_YUY2_RAW,VID_FORMAT_NV12_RAW VID_FORMAT_GREY_RAW.
buf_index	Physical buffer index of the video frame dequeued by the V4L. This needs to be used by MXUVC application to queue back the video frame after processing, in the mxuvc_video_cb_buf_done() function described later.

2.1.7. *wdr_mode_t*

Description:

Enumeration to set the Wide Dynamic Range Mode.

Declaration:

```
typedef enum {
      WDR_AUTO    = 0,
      WDR_MANUAL = 1,
      NUM_WDR
} wdr_mode_t;
```

2.1.8. *exp_set_t*

Description:

Enumeration to set the sensor exposure Mode.

Declaration:

```
typedef enum {
      EXP_AUTO = 0,
      EXP_MANUAL  = 1,
      NUM_EXP
} exp_set_t;
```

2.1.9. *zone_exp_set_t*

Description:

Enumeration to enable/disable sensor zonal exposure.

Declaration:

```
typedef enum {
      ZONE_EXP_DISABLE = 0,
      ZONE_EXP_ENABLE  = 1,
      NUM_ZONE_EXP
} zone_exp_set_t;
```

2.1.10. *noise_filter_mode_t*

Description:

Enumeration to set the noise filter mode for the image processing.

Declaration:

```
typedef enum {
      NF_MODE_AUTO = 0,
      NF_MODE_MANUAL = 1,
      NUM_NF
} noise_filter_mode_t;
```

2.1.11. *white_balance_mode_t*

Description:

Enumeration to set the white balance mode in the sensor.

Declaration:

```
typedef enum {
      WB_MODE_AUTO = 0,
      WB_MODE_MANUAL = 1,
      NUM_WB
}white_balance_mode_t;
```

2.1.12. *zone_wb_set_t*

Description:

Enumeration to enable/disable sensor zonal white balance.

Declaration:

```
typedef enum {
      ZONE_WB_DISABLE = 0,
      ZONE_WB_ENABLE  = 1,
      NUM_ZONE_EXP
} zone_wb_set_t;
```

2.1.13. *pwr_line_freq_mode_t*

```
typedef enum {
      PWR_LINE_FREQ_MODE_DISABLE = 0,
      PWR_LINE_FREQ_MODE_50HZ = 1,
      PWR_LINE_FREQ_MODE_60HZ = 2
}pwr_line_freq_mode_t;
```

2.1.14. *crop_info_t*

Description:

Enumeration to enable/disable zonal exposure.

Declaration:

```
typedef struct {
      uint16_t enable;
      uint16_t width;
      uint16_t height;
      uint16_t x;
      uint16_t y;
} crop_info_t;
```

Variables:

`Enable`	Enable/Disable cropping. Default: 0, Min: 0, Max : 1
`Width`	Width to be cropped from the image. Default: 640, Min: 16, Max : 1920.
`Height`	Height to be cropped from the image. Default: 480, Min: 16, Max: 1080.
`X`	X offset from which the image is cropped. Default: 0, Min: 0, Max: 1920.
`Y`	Y Offset from which the image is cropped. Default: 0, Min: 0, Max: 1080.

2.2. Initializing video subsystem

Initialization of video subsystem involves

- Initializing the video interface.
- Configuring the video channels.
- Registering the video call-back functions.

The API's used for this are described below

2.2.1. Initializing the video interface.

2.2.1.1. *mxuvc_video_init*()

Description:

This API is used to initialize the Linux video interface on the Host system.

Declaration:

```
int mxuvc_video_init(const char *backend, const char *options);
```

Arguments:

- `backend` : string representing the video backend. It is `"v4l2"` in this case.
- `options` : semi-colon separated list of options. Following 2 options can be specified with this parameter
    - `dev_offset`: starting offset of the Geo camera video device node. If the camera is recognized as `/dev/video0` and `/dev/video1` then `dev_offset` should be equal to 0. If it is recognized as `/dev/video1` and `/dev/video2` (because there is another camera in `/dev/video0`) then `dev_offset` should be equal to 1, etc. This option is not mandatory. The default value is 0.
    - `dev_offset_secondary`: offset of second video node (for raw capture). If the camera is recognized as `/dev/video1` and `/dev/video2` then `dev_offset_secondary` should be 2. This option can be used in cases where device numbers are not fixed and may change when devices are reconnected. This option is not mandatory and will default to `dev_offset+1`.
    - `v4l_buffers`: Number of memory mapped buffers for each device. This option is also not mandatory. The default value is 8.

Return Value:

0 on Success, -1 on Failure.

Example Usage:

```
mxuvc_video_init("v4l2", "dev_offset=0,v4l_buffers=16");
or
mxuvc_video_init("v4l2", "");
```

2.2.2. Configuring the video channels

2.2.2.1. *mxuvc_video_get_channel_count*()

Description:

This API is used to get the number of video channels supported by the Geo Camera. The number of video channels supported by the camera is configuration dependent.

Declaration:

```
int mxuvc_video_get_channel_count(int *count);
```

Arguments:

`count`: number of video channels supported. This parameter is returned from the camera based on the configuration in which it is running.

Return Value:

0 on Success, -1 on Failure.

2.2.2.2. mxuvc_video_get_channel_info()

Description:

This API gets the information regarding the video parameters set on the channel in the Camera.

Declaration:

```
int mxuvc_video_get_channel_info(
video_channel_t ch,video_channel_info_t *info);
```

Arguments:

ch:	video channel from which the information is needed.
info:	pointer to the video channel information structure returned from the camera

Return Value:

0 on Success, -1 on Failure.

2.2.3. Registering the call Back function

2.2.3.1. mxuvc_video_register_cb()

Description:

This API is used to register the call back function with the MXUVC. MXUVC calls this user function when the video data is available from the camera. The Application can use this function to process the video data received from the camera.

Declaration:

```
int mxuvc_video_register_cb(video_channel_t ch,
mxuvc_video_cb_t  func, void *user_data);
```

Arguments:

ch:	video channel on which the call back function is registered.
func	Pointer to the call back function to be registered with mxuvc, described below.
user_data	Pointer to mxuvc user handle to be used for further processing, when the Call back function is called by the mxuvc.

Return Value:

0 on Success, -1 on Failure.

An example declaration of the call back function is as below.

```
void example_cb(unsigned char *buffer, unsigned int
size,video_info_t info, void *user_data)
```

Arguments:

buffer	Pointer to the video data received from the camera.
size	Size of the video data received from the camera.
user_data	Pointer to mxuvc user handle set when the call back function is registered.
info	Information regarding the video frame, received from the camera.

2.3. Start/Stop capturing video on the channels

2.3.1. *mxuvc_video_start*()

Description:

This API is used to start the capture of video data from the specified channel in the camera.

Declaration:

```
int mxuvc_video_start(video_channel_t ch);
```

Arguments:

`ch`: video channel on which the video capture needs to be started.

Return Value:

0 on Success, -1 on Failure.

2.3.2. *mxuvc_video_alive*()

Description:

This API is used to check if the camera is active. It can be periodically called to check the availability of the camera. This function returns an error (a negative value) in the following conditions:

- the camera has been unplugged
- the camera or the USB host is no longer responding

Declaration:

```
int mxuvc_video_alive(void);
```

Return Value:

0 on Success, -1 on Failure.

2.3.3. *mxuvc_video_cb_buf_done*()

Description:

This API is used to return back the buffer, received in the call back, back to the mxuvc, when the user has completed processing the video data.

Declaration:

```
int mxuvc_video_cb_buf_done(video_channel_t ch, int buf_index);
```

Arguments:

`ch`:	video channel on which the buffer needs to be returned back to the mxuvc.
`buf_index`	Physical Buffer Index returned as part video information structure (2.1.6) in the call back function. This is used to queue the buffer back to the V4L in the mxuvc.

2.3.4. *mxuvc_video_stop*()

Description:

This API is used to stop the capture of video data from a channel in the camera.

Declaration:

```
int mxuvc_video_stop(video_channel_t ch);
```

Arguments:

`ch:` video channel on which the video capture needs to be stopped.

Return Value:

0 on Success, -1 on Failure.

2.4. Changing the video channel parameters

2.4.1. *mxuvc_video_force_iframe*()

Description:

This API is used to force an I frame in the specified video channel. Applicable only for `VID_FORMAT_H264_RAW, VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.`

Declaration:

```
int mxuvc_video_force_iframe(video_channel_t ch);
```

Arguments:

`ch:` Video channel on which I frame needs to be forced.

Return Value:

0 on Success, -1 on Failure.

2.4.2. *mxuvc_video_get_format*()

Description:

This API is used to get the format of the video data on the specified channel.

Declaration:

```
int mxuvc_video_get_format(video_channel_t ch, video_format_t *fmt);
```

Arguments:

`ch:` video channel on which the format is queried.
`fmt:` format of the video in the specified channel

Return Value:

0 on Success, -1 on Failure.

2.4.3. *mxuvc_video_set_format*()

Description:

Not Supported.

2.4.4. *mxuvc_video_(get/set)_resolution*()

Description:

This API is used to get the resolution of the video data on the specified channel.

Declaration:

```
int mxuvc_video_get_resolution(video_channel_t ch,
uint16_t *width, uint16_t *height);
int mxuvc_video_set_resolution(video_channel_t ch,
uint16_t width, uint16_t height);
```

Arguments:

`ch:` video channel on which the resolution is queried.
`width:` width of the video in the specified channel.
`height:` height of the video in the specified channel.

Return Value:

0 on Success, -1 on Failure.

2.4.5. *mxuvc_video_(get/set)_crop*()

Description:

This API is used to get/set the video crop paramters on the specified channel.

Declaration:

```
int mxuvc_video_set_crop(video_channel_t ch, crop_info_t *info);
int mxuvc_video_get_crop(video_channel_t ch, crop_info_t *info);
```

Arguments:

`ch:` video channel on which the resolution is queried.
`info:` video crop parameters to be get/set on the channel,

Return Value:

0 on Success, -1 on Failure.

API's to get and set some of the video channel parameters follow same syntax as describe below.

Declaration:

```
int   mxuvc_video_get_<param>(video_channel_t ch, uint32_t* param_val);
```

Arguments:

`ch:` Video from which the parameter is queried.
`param_val:` Pointer to parameter value, updated by the camera.

Return Value:

0 on Success, -1 on Failure.

Declaration:

```
int   mxuvc_video_set_<param>(video_channel_t ch, uint32_t param_val);
```

Arguments:

`ch:`	Video channel on which the parameter is set.
`param_val:`	Parameter value, to be set in the camera.

Return Value:

0 on Success, -1 on Failure.

API's for all the parameters and their expected ranges are described below.

2.4.6. *mxuvc_video_(get/set)_framerate*()

Description:

Gets or sets the framerate on the video channel specified. Not applicable for `VID_FORMAT_YUY2_RAW,VID_FORMAT_NV12_RAW,VID_FORMAT_GREY_RAW.`

Typical Range: 1 to 30.

2.4.7. *mxuvc_video_(get/set)_goplen*()

Description:

Gets or sets the GOP length on the video channel specified. Applicable only for `VID_FORMAT_H264_RAW, VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.`

Typical Range: 0 to Max Integer (2147483647) .

2.4.8. *mxuvc_video_(get/set)_bitrate*()

Description:

Gets or sets the bitrate on the video channel specified. Not applicable for `VID_FORMAT_YUY2_RAW,VID_FORMAT_NV12_RAW,VID_FORMAT_GREY_RAW.`

Typical Range: 100000 (100 kbps) to 2000000 (2 Mbps)

2.4.9. *mxuvc_video_(get/set)_profile*()

Description:

Gets or sets the H264 profile on the video channel specified. Applicable only for `VID_FORMAT_H264_RAW, VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.`

Values Supported:

PROFILE_BASELINE, PROFILE_MAIN, PROFILE_HIGH. (See 2.1.2)

2.4.10. *mxuvc_video_(get/set)_maxnal*()

Description:

Gets or sets the maximum size of the NAL unit received from the camera. If this parameter is set to 0 the camera sends variable size NAL units. If it is set to finite value the camera splits the frame into multiple equal size NAL units with the maximum size equal to the parameter value. Applicable only for `VID_FORMAT_H264_RAW, VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.`

Typical Range: 0 to 2000.

2.4.11. *mxuvc_video_(get/set)_compression_quality*()

Description:

Gets or Sets the compression quality in terms of image quantization parameter (QP) Applicable only for `VID_FORMAT_MJPEG_RAW`.

Typical Range: 0 to 10000.

2.4.12. *mxuvc_video_(get/set)_avc_level*()

Description:

Gets or sets AVC/H264 level on the video channel. Applicable only for `VID_FORMAT_H264_RAW, VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.`

Typical Range: 10 to 52.

2.4.13. *mxuvc_video_(get/set)_vui*()

Description:

Enables/Disables VUI NAL Unit in the H264 video channel.
Applicable only for `VID_FORMAT_H264_RAW, VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.`

Values: 0 – Disable, 1- Enable.

2.4.14. *mxuvc_video_(get/set)_pict_timing*()

Description:

Enables/Disables picture timing NAL in the H264 video channel.
Applicable only for `VID_FORMAT_H264_RAW, VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.`

Values: 0 – Disable, 1- Enable.

2.4.15. *mxuvc_video_(get/set)_gop_hierarchy_level*()

Description:

Gets or sets the GOP Hierarchy level Applicable only for `VID_FORMAT_H264_RAW,VID_FORMAT_H264_TS,VID_FORMAT_H264_AAC_TS.`

Typical Renge: 0 to 4.

2.4.16. *mxuvc_video_(get/set)_max_framesize*()

Description:

This API is set the maximum size of the I frame in the specified video channel. Applicable only for `VID_FORMAT_H264_RAW, VID_FORMAT_H264_TS, VID_FORMAT_H264_AAC_TS.`

Typical Renge: 0 to 64000.

2.5. Changing the sensor/image parameters

MXUVC also exposes API's to set sensor/Image parameters on the camera.Some of these function follow same syntax as described below.

Declaration:

```
int   mxuvc_video_get_<param>(video_channel_t ch,
uint32_t* param_val);
```

Arguments:

`ch:`	video channel on which the parameter is set.
`param_val:`	Pointer to parameter value, updated by the camera.

Declaration:

```
int   mxuvc_video_set_<param>(video_channel_t ch,
uint32_t param_val);
```

Return Value:

0 on Success, -1 on Failure.

Arguments:

`ch:`	video channel on which the parameter is set.
`param_val:`	Parameter value, to be set in the camera.

Return Value:

0 on Success, -1 on Failure.

API's for all the parameters and their typical ranges are listed below.

2.5.1. *mxuvc_video_(get/set)_flip_horizontal*()

Description:

Gets/Sets the image contrast

Values: 1 Enable, 0 Disable

2.5.2. *mxuvc_video_(get/set)_flip_vertical*()

Description:

Gets/Sets the image contrast

Values: 1 Enable, 0 Disable

2.5.3. *mxuvc_video_(get/set)_contrast*()

Description:

Gets/Sets the image contrast

Typical Range: 0 to 200.

2.5.4. *mxuvc_video_(get/set)_zoom*()

Description:

Gets/Sets the image zoom step

Typical Range: 0 to 100.

2.5.5. *mxuvc_video_(get/set)_pan*()

Description:

Gets/Sets the image pan level.

Typical Range: pan: -648000 to 648000

2.5.6. *mxuvc_video_(get/set)_tilt*()

Description:

Gets/Sets the image tilt level

Typical Range: tilt: -648000 to 648000

2.5.7. *mxuvc_video_(get/set)_pantilt*()

Description:

Gets/Sets the image pan and tilt level

Typical Range: pan: -648000 to 648000, tilt: -648000 to 648000

2.5.8. *mxuvc_video_(get/set)_brightness*()

Description:

Gets/Sets the image Brightness.

Typical Range: -255 to 255.

2.5.9. *mxuvc_video_(get/set)_hue*()

Description:

Gets/Sets the image Hue.

Typical Rangec: -18000 to 18000.

2.5.10. *mxuvc_video_(get/set)_gamma*()

Description:

Gets/Sets the image gamma.

Typical Range: 100 to 300.

2.5.11. *mxuvc_video_(get/set)_saturation*()

Description:

Gets/Sets the image Saturation.

Typical Range: 0 to 200.

2.5.12. *mxuvc_video_(get/set)_gain*()

Description:

Gets/Sets the image Gain.

Typical Range: 1 to 100.

2.5.13. *mxuvc_video_(get/set)_sharpness*()

Description:

Gets/Sets the image sharpness.

Typical Range: 0 to 100.

2.5.14. *mxuvc_video_(get/set)_max_analog_gain*()

Description:

Controls the maximum sensor analog gain in auto exposure algorithm.

Typical Range: 0 to 15.

2.5.15. *mxuvc_video_(get/set)_histogram_eq*()

Description:

Disables/enables Image histogram equalization.

Typical Range: 0 : Disable 1 : enable

2.5.16. *mxuvc_video_(get/set)_sharpen_filter*()

Description:

Gets/Sets strength of the Image sharpening filter.

Typical Range: 0,1 and 2. 2 being the strongest.

2.5.17. *mxuvc_video_(get/set)_min_exp_framerate*()

Description:

Gets/Sets minimum exposure .framerate

Typical Range: 0 to 30.

2.5.18. mxuvc_video_(get/set)_tf_strength()

Description:

Gets/Sets the temporal filter strength, Value 0 disables the temporal filter. Value 1 to 7 sets the filter strength.

Typical Range: 0 to 7.

2.5.19. *mxuvc_video_(get/set)_gain_multiplier* ()

Description:

Controls the auto exposure algorithm to adjust the sensor analog gain and exposure based on different lighting conditions.

Typical Range: 0 to 256

2.5.20. *mxuvc_video_(get/set)_exp*()

Description:

This API is used to get or set the exposure mode (auto or manual) and the exposure time in the manual mode.

Declaration:

```
int mxuvc_video_set_exp(video_channel_t ch
,exp_set_t sel, uint16_t value);
int mxuvc_video_get_exp(video_channel_t ch,
exp_set_t *sel, uint16_t *value);
```

Arguments:

`ch:` active video channel from which the parameter is set/got.
`sel:` selects to exposure mode (Auto or Manual).
`value:` exposure time value in the Manual Mode - Range 0 to 255.

2.5.21. *mxuvc_video_(get/set)_nf*()

Description:

This API is used to get or set the noise filter mode (auto or manual) and the noise filter strength in the manual mode.

Declaration:

```
int mxuvc_video_set_nf(video_channel_t ch,
noise_filter_mode_t sel,      uint16_t value);
int mxuvc_video_get_nf(video_channel_t ch,
noise_filter_mode_t *sel,     uint16_t *value);
```

Arguments:

`ch:` active video channel from which the parameter is set/got.
`sel:` selects to noise filter mode (Auto or Manual),
`value:` noise filter strength in the Manual Mode - Range 0 to 100.

2.5.22. *mxuvc_video_(get/set)_wb*()

Description:

This API is used to get or set the white balance mode (auto or manual) and the white balance temperature in the manual mode.

Declaration:

```
int mxuvc_video_set_wb(video_channel_t ch,
white_balance_mode_t sel, uint16_t value);
int mxuvc_video_get_wb(video_channel_t ch,
white_balance_mode_t *sel, uint16_t *value);
```

Arguments:

`ch:` active video channel from which the parameter is set/got.
`sel:` selects to white balance mode (Auto or Manual),
`value:` white balance temperature in the manual Mode - Range 2800 to 6500.

2.5.23. *mxuvc_video_(get/set)_wdr*()

Description:

This API is used to get or set the camera wide dynamic range mode (WDR) (auto or manual) and the WDR control intensity in the manual mode.

Declaration:

```
int mxuvc_video_set_wdr(video_channel_t ch,
wdr_mode_t mode, uint8_t value);
int mxuvc_video_get_wdr(video_channel_t ch,
wdr_mode_t *mode, uint8_t *value);
```

Arguments:

`ch:` active video channel from which the parameter is set/got.
`mode:` selects to WDR mode (Auto or Manual).
`value:` WDR control intensity value in the Manual Mode - Range 0 to 255.

2.5.24. *mxuvc_video_(get/set)_zone_*exp()

Description:

This API is used to enable/disable zonal exposure, get/set the exposure zone.

Declaration:

```
int mxuvc_video_set_zone_exp(video_channel_t ch,
zone_exp_set_t sel,     uint16_t value);
int mxuvc_video_get_zone_exp(video_channel_t ch,
zone_exp_set_t *sel, uint16_t *value);
```

Arguments:

`ch:` active video channel from which the parameter is set/got.
`sel:` enable/disable zonal exposure.
`value:` exposure zone value Range 0 to 62.

2.5.25. *mxuvc_video_(get/set)_zone_*wb()

Description:

This API is used to enable/disable zonal white balance and get/set the white balance zone.

Declaration:

```
int mxuvc_video_set_zone_exp(video_channel_t ch,
zone_wb_set_t sel, uint16_t value);
int mxuvc_video_get_zone_exp(video_channel_t ch,
zone_wb_set_t *sel, uint16_t *value);
```

Arguments:

`ch:` active video channel from which the parameter is set/got.
`sel:` enable/disable zonal white balance.
`value:` white balnce zone value Range 0 to 63.

2.5.26. *mxuvc_video_(get/set)_pwr_line_freq*()

Description:

This API is used to select the power line frequency of the operating region. Sensor exposure value under the auto-exposure algorithm will be adjusted to avoid flickering caused by power level oscillation. 0 disables this function, and the values of 1 and 2 represents 50 and 60Hz power line frequency, respectively.

Declaration:

```
int mxuvc_video_set_pwr_line_freq(
video_channel_t ch,pwr_line_freq_mode_t mode);
int mxuvc_video_get_pwr_line_freq(
video_channel_t ch,pwr_line_freq_mode_t *mode);
```

Arguments:

`ch:` active video channel from which the parameter is set/got.
`mode:` value indicating power line frequency mode.
0 – disable, 1 – 50Hz, 2 – 60Hz .

2.6. Freeing the resources

2.6.1. mxuvc_video_deinit()

Description:

This API is used to free all the memory allocated by the MXUVC. This function automatically calls `mxuvc_video_stop()` if the video has not been stopped yet, it is therefore not necessary to explicitly call `mxuvc_video_stop()` before calling `mxuvc_video_deinit()`.

Declaration:

```
int mxuvc_video_deinit();
```

Return Value:

0 on Success, -1 on Failure.

3. MXUVC AUDIO SUBSYSTEM

MXUVC Audio subsystem is used to intialize the Geo camera Audio Interface, configuring the audio parameters and start capturing the audio data for further processting. MXUVC as an API supports capture of multiple channels of audio data of different sampling frequency and format. However in the current Geo Camera product configuration we support only one channel of Audio Data which is of format AAC. The Data Structures and the APIs are explained in the below sections

3.1. Data Structures

3.1.1. *audio_format_t*

Description:

Enumeration to indicate the format type supported by the channel.

Declaration:

```
typedef enum {
      AUD_FORMAT_PCM_RAW  = 0,
      AUD_FORMAT_AAC_RAW  = 1,
      NUM_AUD_FORMAT
} audio_format_t;
```

3.1.2. *audio_channel_t*

Description:

Enumeration to indicate the MXUVC Audio Channel.

Declaration:

```
typedef enum {
      AUD_CH1 = 0,
      AUD_CH2,
      NUM_AUDIO_CHANNELS
} audio_channel_t;
```

3.1.3. *audio_codec_type_t*

Description:

Enumeration to indicate the codec type used for encoding in the channel.

Declaration:

```
typedef enum {
      AUDIO_CODEC_TYPE_AAC = 0x1,
      AUDIO_CODEC_TYPE_QAC = 0x1,
      AUDIO_CODEC_TYPE_QPCM = 0x3,
      AUDIO_CODEC_TYPE_Q711 = 0x9,
      AUDIO_CODEC_TYPE_Q722 = 0xa,
      AUDIO_CODEC_TYPE_Q726 = 0xb,
      AUDIO_CODEC_TYPE_Q728 = 0xc,
      AUDIO_CODEC_TYPE_AMRNB = 0x10,
} audio_codec_type_t;
```

3.1.4. *audio_params_t*

Description:

Structure containing the parameters of the audio data received from the channel in the call-back.

Declaration:

```
typedef struct
{
      long long               timestamp;
      int                     framesize;
      int                     samplefreq;
      int                     channelno;
      int                     audioobjtype;
      unsigned char           *dataptr;
      audio_codec_type_t      audiocodectype;
} audio_params_t;
```

Variables:

`timestamp`	audio timestamp in terms of ticks of 90khz clock where each tick corresponds to 1/(90 * 1000) sec or 1/90 ms
`framesize`	size of the audio frame receieved.
`samplefreq`	sampling frequency at which the audio frame is captured.
`channelno`	number of audio channels captured by the microphone and/or encoded.
`audioobjtype`	AAC Audio Object Type with which the Audio stream is encoded. This is useful to construct the ADTS Header. Ignore in case of PCM.
`dataptr`	pointer to audio frame data.
`audiocodectype`	encoded type of the Audio Stream.

3.2. Initializing audio subsystem.

3.2.1. *mxuvc_audio_init*()

Description:

This API is used to initialize the Linux audio interface on the Host system.

Declaration:

```
int mxuvc_audio_init(const char *backend, const char *options);
```

Arguments:

- `backend` : string representing the audio backend. It is `"alsa"` in this case.
- `options` : semi-colon separated list of options. Following options can be specified with this parameter.
    - `device`: This is a mandatory parameter used to specify the ALSA name of the Geo Camera, typically MAX64380.
    - `audio_sampling_rate`: This parameter is used to specify the sampling rate at which the audio needs to be captured. It is not a mandatory parameter. Default value is 24khz.

- audio_duration_ms: This parameter is used to set the capture period size in ms. It is not a mandatory parameter. Default value is 10ms.

Return Value:

0 on Success, -1 on Failure.

Example Usage:

```
mxuvc_audio_init("alsa","\
                 device = MAX64380;\
               audio_sampling_rate = 24000; \
               audio_duration_ms = 10");
or
mxuvc_audio_init("alsa", "device = MAX64380");
```

3.3. Registering the call back function

3.3.1. mxuvc_audio_register_cb()

Description:

This API is used to register the call back function with the MXUVC. MXUVC calls this user function when the audio data is available from the camera on the channel specified. The Application can use this function to process the audio data received from the camera.

Declaration:

```
int mxuvc_audio_register_cb(audio_channel_t ch,
mxuvc_audio_cb_t func,void *user_data);
```

Arguments:

ch:	audio channel on which the call back function is registered.
func	Pointer to the call back function to be registered with mxuvc, described below.
user_data	Pointer to mxuvc user handle to be used for further processing, when the Call back function is called by the mxuvc.

Return Value:

0 on Success, -1 on Failure.

An example declaration of the call back function is as below.

```
void example_cb(unsigned char *buffer,unsigned int size,
audio_format_t format, uint64_t ts,
void *user_data, audio_params_t *param);
```

Arguments:

buffer	Pointer to the audio data received from the camera.
size	Size of the audio data received from the camera.
user_data	Pointer to mxuvc user handle set when the call back function is registered.
format	format of the audio frame recieved.
ts	audio timestamp in terms of ticks of 90khz clock where each tick corresponds to 1/(90 * 1000) sec or 1/90 ms
param	Pointer to the audio parameters in the current received from the audio channel in the call back function.

3.4. Start/Stop capturing audio on the channels

3.4.1. *mxuvc_audio_start*()

Description:

This API is used to start the capture of audio data from a specified channel in the camera.

Declaration:

```
int mxuvc_audio_start(audio_channel_t ch);
```

Arguments:

`ch:` audio channel on which the audio capture needs to be started.

Return Value:

0 on Success, -1 on Failure.

3.4.2. *mxuvc_audio_alive*()

Description:

This API is used to check if the camera is active. It can be periodically called to check the availability of the camera. This function returns an error (a negative value) in the following conditions:

- the camera has been unplugged
- the camera or the USB host is no longer responding

Declaration:

```
int mxuvc_audio_alive(void);
```

Return Value:

0 on Success, -1 on Failure.

3.4.3. *mxuvc_audio_stop*()

Description:

This API is used to stop the capture of audio data from a specified channel in the camera.

Declaration:

```
int mxuvc_audio_stop(audio_channel_t ch);
```

Arguments:

`ch:` audio channel on which the audio capture needs to be started.

Return Value:

0 on Success, -1 on Failure.

3.5. Changing audio parameters

3.5.1. *mxuvc_audio_set_samplerate*()

Description:

This API is used to set the audio sampling rate on the channel .

Declaration:

```
int   mxuvc_audio_set_samplerate(audio_channel_t, int
samplingFr);
```

Arguments:

`ch:`	audio channel on which the audio capture needs to be started.
`samplingFr:`	Sampling frequency to be set.

Return Value:

0 on Success, -1 on Failure.

3.5.2. *mxuvc_audio_set_mic_gain*()

Description:

This API is used to set the microphone gain.

Declaration:

```
int   mxuvc_audio_set_mic_gain(int level);
```

Arguments:

`level:`	microphone gain level. Range is from 0 to 100.

Return Value:

0 on Success, -1 on Failure.

3.5.3. *mxuvc_audio_set_mic_mute*()

Description:

This API is used to mute or unmute the micro phone.

Declaration:

```
int   mxuvc_audio_set_mic_mute(int bMute);
```

Arguments:

`bMute:`	microphone mute value. 0 - mute 1 - un mute

Return Value:

0 on Success, -1 on Failure.

3.5.4. *mxuvc_audio_set_bitrate*()

Description:

This API is used to set the bitrate of the Audio.

Declaration:

```
int   mxuvc_audio_set_bitrate(audio_channel_t ch, int bitrate);
```

Arguments:

`ch:`	audio channel on which the bitrate needs to be set.
`bitrate:`	bitrate to be set. (Range 16000 to 320000)

Return Value:

0 on Success, -1 on Failure.

3.6. Freeing the resources

3.6.1. mxuvc_audio_deinit()

Description:

This API is used to free all the memory allocated by the MXUVC. This function automatically calls `mxuvc_audio_stop()` if the video has not been stopped yet, it is therefore not necessary to explicitly call `mxuvc_audio_stop()` before calling `mxuvc_audio_deinit()`.

Declaration:

```
int mxuvc_audio_deinit();
```

Return Value:

0 on Success, -1 on Failure.

4. MXUVC ALERT SUBSYSTEM

MXUVC Alert subsystem is used to configure and receive Motion and Audio Alerts from the Geo Camera. The Data Structures and the APIs are explained in the below sections

4.1. Data Structures

4.1.1. *alert_motion_data*

Description:

Structure used to get the motion statistics from the camera required for triggering the motion alert.

Declaration:

```
typedef struct {
      int         reg_id;
      uint8_t     transition;
      uint16_t    mbs_in_motion;
      uint16_t    avg_motion;
      uint32_t    PTS_high;
      uint32_t    PTS_low;
}alert_motion_data;
```

Variables:

reg_id:	region Id of the rectangle on which the motion alert is triggered
transition :	indicates the current motion alert transition. Static to motion or vice-versa.Values : a) 1 indicates transistion from static to motion. b) 0 indicates transistion from motion to static.
mbs_in_motion:	number of macroblocks which are in motion.
avg_motion:	average motion in the rectangle.
PTS_high:	Higher 32 bits of the Time Stamp of motion.
PTS_low:	Lower 32 bits of the Time Stamp of motion.

4.2. Initializing alert subsystem.

4.2.1. *mxuvc_alert_init*()

Description:

This API is used to initialize the Alert sub system.

Declaration:

```
int mxuvc_alert_init(void);
```

Return Value:

0 on Success, -1 on Failure.

4.3. Interface functions related to motion alerts

4.3.1. *mxuvc_alert_motion_enable*()

Description:

This API is used to register the call back function to enable receiving the motion alert statistics.

Declaration:

```
int  mxuvc_alert_motion_enable(void(*callback)(alert_motion_data*,  void
*data), void *data);
```

Arguments:

callback	Pointer to the call-back function, to receive the motion alert statistics.
data	Pointer to mxuvc user handle to be used for processing, when the Call back function is called by the mxuvc.

An example declaration of the call back function is as below.

```
void example_cb(alert_motion_data* alert_data, void *user_data);
```

Arguments:

alert_data	Pointer to the motion alert statistics.
user_data	Pointer to mxuvc user handle set when the call back function is registered.

Return Value:

0 on Success, -1 on Failure.

4.3.2. mxuvc_alert_motion_add_region()

Description:

This API is used to add region of interest for the motion alert. Maximum number of regions supported is 16.

Declaration:

```
int mxuvc_alert_motion_add_region(int reg_id,
int x_offt, int  y_offt, int width, int height);
```

Arguments:

reg_id:	region Id of the rectangle on which the motion alert needs to be monitored.
x_offt	x offset of the rectangle.
y_offt	y offset of the rectangle.
width	width of the rectangle in terms of macroblocks. i.e image_width/16
height	height of the rectangle in terms of macroblocks. i.e image_height/16

Return Value:

0 on Success, -1 on Failure.

4.3.3. mxuvc_alert_motion_set_sensitivity_and_level()

Description:

This API is used to set the motion sensitivity and threshold on the region of interest.

Declaration:

```
int mxuvc_alert_motion_set_sensitivity_and_level
(int reg_id,int sensitivity,int level);
```

Arguments:

reg_id:	region Id of the rectangle on which the motion alert needs to be monitored.
sensitivity	threshold for motion detection for a macroblock.Valid range for this variable is any integer between 16 to 200. Lower the value, more the motion alerts triggered.
level	level of motion in the region in terms of percent of macroblocks in the region of interest should be in motion for alert to be generated. Valid range for this variable is any integer between 1 to 100.

Return Value:

0 on Success, -1 on Failure.

Example Usage:

```
/*set motion region in region id zero and covering a 720p frame. */
reg_id = 0;
x_offt = 0;
y_offt = 0;
width = 80;
height = 45;
sensitivity = 50;
level = 100;
ret = mxuvc_alert_motion_add_region(reg_id, x_offt, y_offt, width,
                height);
ret = mxuvc_alert_motion_set_sensitivity_and_level(
sensitivity, level);
```

4.3.4. mxuvc_alert_motion_remove_region()

Description:

This API is used to remove the added region from region of interest for motion alert.

Declaration:

```
int mxuvc_alert_motion_remove_region(int reg_id);
```

Arguments:

reg_id:	region Id of the rectangle on which the motion alert needs to be monitored.

Return Value:

0 on Success, -1 on Failure.

Example Usage:

```
reg_id = 0;
```

```
ret = mxuvc_alert_motion_remove_region(reg_id);
```

4.3.5. *mxuvc_alert_motion_disable*()

Description:

This API is used is used to disable motion alert.

Declaration:

```
int mxuvc_alert_motion_disable(void);
```

Return Value:

0 on Success, -1 on Failure.

4.4. Interface functions related to audio alerts

4.4.1. *mxuvc_alert_audio_enable*()

Description:

This API is used to register the call back function to enable receiving the audio alerts.

Declaration:

```
int mxuvc_alert_audio_enable(
void (*callback)(unsigned int audioIntensityDB, void* data)
, void* data);
```

Arguments:

`callback`	Pointer to the call-back function, to receive the audio alerts.
`data`	Pointer to mxuvc user handle to be used for processing, when the Call back function is called by the mxuvc.

An example declaration of the call-back function is as below.

```
void example_cb(void *user_data);
```

Arguments:

`user_data`	Pointer to mxuvc user handle set when the call back function is registered.

Return Value:

0 on Success, -1 on Failure.

4.4.2. *mxuvc_alert_audio_set_threshold*()

Description:

This API is used to set audio intensity threshold in terms of decibels, above which the audio alert should be triggered.

Description:

```
int mxuvc_alert_audio_set_threshold(
      unsigned int audioThresholdDB);
```

Arguments:

`audioIntensityDB`	Information passed by the camera indicating current intensity of the audio in terms decibels, in the monitoring environment as perceived by the microphone.

Return Value:

0 on Success, -1 on Failure.

4.4.3. *mxuvc_get_current_audio_intensity*()

Description:

This API is used to get the current intensity of the audio in the monitoring environment, in terms of decibels, perceived by the microphone.

Description:

```
int mxuvc_get_current_audio_intensity
(unsigned int* audioIntensityDB);
```

Arguments:

`audioIntensityDB`	Information passed by the camera indicating current intensity of the audio in terms decibels, in the monitoring environment as perceived by the microphone.

Return Value:

0 on Success, -1 on Failure.

4.4.4. *mxuvc_alert_audio_disable*()

Description:

This API is used is used to disable audio alert.

Declaration:

```
int mxuvc_alert_audio_disable(void);
```

Return Value:

0 on Success, -1 on Failure.

4.5. Freeing the resources

4.5.1. *mxuvc_alert_deinit*()

Description:

This API is used to release all the resources allocated by the alert sub system.

Declaration:

```
int mxuvc_alert_deinit(void);
```

Return Value:

0 on Success, -1 on Failure.

5. MXUVC BURN-IN PLUG-IN

MXUVC Burn-In Plug-in is used to overlay customary text real-time in the captured video. One of the use-case of the feature is to insert the current time information, during the video capture. The requirements of this interface are described below

- **Language**: The text supported for Burn-In is English only
- **Size**: The size of the font supported could be 16x16, 32x32, or 8x8. 8x8 may not be suitable for HD resolution.
- **Files**: By default standard Font files are released for this feature. However the user would be able to use custom fonts.
- **Download:** The user should be allowed to download the font binary over the USB as a part of camera initialization. The Font binary is not stored in the firmware. However space is allocated for maximum allowable font size (32 x 32) in the firmware.
- **Posistion:** The user should be allowed to specify a position X and Y of the video at which he wants to overlay the text, where X and Y are constrained to below in the firmware.

  X = (X + (TILE_WIDTH – 1))/TILE_WIDTH

  Max value of X calculation

  tempX = (image_width – (text_length * font_width))

  X = (tempX + (TILE_WIDTH – 1))/ TILE_WIDTH

  Y = (Y + (TILE_HEIGHT – 1))/TILE_HEIGHT

  Max value of Y calculation

  For font size 16X16 and 8x8

  Y = image_height - TILE_HEIGHT

  For font size 32x32

  Y = image_height - 32

  Where TILE_WIDTH = 64 TILE_HEIGHT = 16 in 128 MB DDR configuration
- **Number:** The user should be allowed to specify upto 3 Burn-In texts at a time at independent positions X and Y. However the length of each text is fixed to a maximum of 24 characters.
- **Update:** The user should be able to change the text which he needs to overlay on the video at a given point of time during capture.
- **Burn-In Time :** When the user uses Burn-In time feature
  1. The time should be displayed in 24Hr Format.
  2. The user should be able to set a start time every now and then. The Raptor firmware then automatically increments and displays the time from the specified start time. By default the firmware sets the start time as 00:00:00.

5.1. Interface functions related to Burn-In

5.1.1. *mxuvc_burnin_init*()

Description:

This API is used to initialize the Burn-In plug-in system. This API decompresses the font binary and is expected to take about 100 miliseconds. This API must be called before starting the Video and Audio Capture.

Declaration:

```
int mxuvc_burnin_init(int font_size, char* file_name);
```

Arguments:

font_size:	size of the Font used to overlay the text in the video. Possible values are 8,16 and 32. Error is returned if the font size is other than these values.
file_name:	Path and filename of the Font binary file to be used for the text. The file provided should be in accordance with the size specified. The text displayed would be garbled other wise.

Return Value:

0 on Success, -1 on Failure.

5.1.2. *mxuvc_burnin_deinit*()

Description:

This API is used to free all the resources allocated by the Burn-In plug-in system.

Declaration:

```
int mxuvc_burnin_deinit();
```

Return Value:

0 on Success, -1 on Failure.

5.1.3. *mxuvc_burnin_add_text*()

Description:

This API is used to add or updated the text of specified text index and at specified posistion. Raptor supports upto 4 texts to be overlayed on the video.

Declaration:

```
int mxuvc_burnin_add_text(int idx,char* str, int length,int X,
      int Y );
```

Arguments:

idx	Index of the text to be used. Maximum of 4 indices are supported by the Raptor. Possible values are 0 to 3.
str	string to be overlayed on the video.
length	length of the string in bytes. Raptor can support maximum string length of 24.
X	X offset in Video at which the string is to be displaced.
Y	Y offset in Video at which the string is to be displaced.

Return Value:

0 on Success, -1 on Failure.

5.1.4. *mxuvc_burnin_remove_text*()

Description:

This API is used to remove the text from the specified text index.

Declaration:

```
int mxuvc_burnin_remove_text(int idx);
```

Arguments:

`idx` Index of the text to be used. Maximum of 4 indices are supported by the Raptor. Possible values are 0 to 3.

Return Value:

0 on Success, -1 on Failure.

5.1.5. *mxuvc_burnin_show_time*()

Description:

This API is used to enable the burn-In of the current time in the video at a specified posistion. The time displayed is in 24 hour HH:MM:SS format By default the start time is 00:00:00. When Frame Number is enabled, the Burn in time is displayed in HH:MM:FF format. Where FF goes from 00 to 99.

Declaration:

```
int mxuvc_burnin_show_time(int x, int y, int frame_num_enable);
```

Arguments:

`X`	X offset in Video at which the string is to be displaced.
`Y`	Y offset in Video at which the string is to be displaced.
`frame_num_enable`	Enable display of Frame Number in the Burn-In time string.

Return Value:

0 on Success, -1 on Failure.

5.1.6. *mxuvc_burnin_hide_time*()

Description:

This API is used to disable the burn-In of the current time in the video.

Declaration:

```
int mxuvc_burnin_hide_time();
```

Return Value:

0 on Success, -1 on Failure.

5.1.7. *mxuvc_burnin_update_time*()

Description:

This API is used to update the start time of the Burn-In timer in the Raptor firmware.

Declaration:

```
int mxuvc_burnin_update_time(int h, int m, int sec);
```

Arguments:

`h`	hour part of start time to be set. Expected value of hour is from 0 to 23
`m`	minute part of the start time to be set. Expected value of hour is from 0 to 60.
`sec`	seconds part of the start time to be set. Expected value of hour is from 0 to 60.

Return Value:

0 on Success, -1 on Failure.

6. MXUVC CUSTOM-CONTROL PLUG-IN

MXUVC Custom-Control Plug-in is used to handle all the custom control(proprietary) commands what is required to support Raptor specific functionalities like Acoustic Echo Cancellor (AEC) etc.

6.1. Interface functions related to Custom-Control

6.1.1. *mxuvc_custom_control_init*()

Description:

This API is used to initialize the custom control plugin.

Declaration:

```
int mxuvc_custom_control_init(void);
```

Return Value:

0 on Success, -1 on Failure.

6.1.2. *mxuvc_custom_control_deinit*()

Description:

This API is used to deallocate all the resources of the custom control plugin.

Declaration:

```
int mxuvc_ custom_control_deinit(void);
```

Return Value:

0 on Success, -1 on Failure.

6.1.3. mxuvc_custom_control_set_audio_codec_samplerate()

Description:

This API is used to set the sample rate of the MAX9860 ADC and the DAC in the Raptor. An Audio resampler is added to the Data Path if the codec sample rate set below is not equal to the capture sample rate set as part of API mxuvc_audio_set_samplerate() or mxuvc_audio_init() above. Also the resampler produces the exact number of samples required for preprocessing filters (AEC eg) or the Encoder (AAC) only when the Conversion Ratio is Integer Multiple. So the supported Combination of the Codec Sample Rate and the Capture Sample Rate are summarised as below. Also to preserve the quality, Up Sampling is prevented in the Resampler, in that case the codec sample rate is set to the capture sample rate.

Codec Sample Rate	Supported Capture Sample Rates		
	8000	16000	24000
48000	Supported	Supported	Supported
44100	Not Supported	Not Supported	Not Supported
32000	Supported	Supported	Not Supported
24000	Supported	Not Supported	Supported Resampling Disabled
16000	Supported	Supported Resampling Disabled	Not Supported

In Case of Not supported. The Codec Sample Rate is made equal to the Capture Sample Rate. The User shpuld take care of this while setting the required capture and codec sample rates.

Declaration:

```
int   mxuvc_custom_control_set_audio_codec_samplerate(unsigned     int samplerate);
```

Arguments:

`samplerate`: sample rate to be set n the MAX9860 DAC and ADC.

Return Value:

0 on Success, -1 on Failure.

6.1.4. mxuvc_custom_control_enable_aec()

Description:

This API is used to set the camera audio capture data path in the AEC mode where-in two Audio Inputs and the Acoustic Echo Cancellor (AEC) preprocessing filter are enabled in the Audio Capture Data Path as below, before giving the samples to encode. One audio input is used to capture the actual microphone data and the second input is used to capture the Far end data which is played through the speaker on the Host.

Audio Filter

Declaration:

```
int   mxuvc_custom_control_enable_aec();
```

Return Value:

0 on Success, -1 on Failure.

6.1.5. mxuvc_custom_control_disable_aec()

Description:

This API is used to set the camera audio capture data path in the Normal mode where-in a single Audio Input is passed through an Audio Alaram preprocessing before giving the samples to encode.

Audio Filter

Declaration:

```
int   mxuvc_custom_control_disable_aec();
```

Return Value:

0 on Success, -1 on Failure.

7. COMPILING MXUVC

Compiling mxuvc results in a static library, **libmxuvc.a**, and a shared library, **libmxuvc.so**. If only one application uses mxuvc API, it is simpler to directly linked in the static library. If more than one application uses the API then space and memory can be spared by using the shared library.

To compile, go to mxuvc/ directory and type:
Cross compiling mxuvc for a specific platform requires the Makefile to be modified to make it point to the toolchain compiler, libraries and header files.

```
# make VIDEO=v4l2 AUDIO=alsa
```

8. EXAMPLE CODE

The mxuvc/examples/capture-mux directory contains various applications to demostrate use of mxuvc API's for the multi-channel video and audio Application.

The mxuvc/examples/alert directory contains applications to demonstrate the alert handling during capture.

8.1. Video Capture

8.1.1. Capture multi channel video

The example `capture_mux.c` demonstrates multi-channel video only capture.
The programming flow of the application is as below.

- Initialize the video interface.
- Get the number of video channels supported by the camera.
- Register the call back function with the channel number as the user parameter to know from which the channel the video data is received.
- Gets and sets various channel paramters.
- Start the video capture on all the all the channels for a duration in secs specified by the first argument to the application, if not specified it is default to 15 secs.
- Sets various channel paramters on the fly.
- The video data received in the call back is written into different files specific to the channel based on the channel parameter received in the call-back.
- Stop the video capture on all the all the channels.
- Free the resources allocated.

8.1.2. Capture multi channel video and audio.

The example `capture_avmux.c` demonstrates multi-channel video and AAC audio capture.
The programming flow of the application is as below.

- Initialize the audio interface.
- Set the Audio format and the sample rate.
- Register the audio call back function to receive audio data.
- Set the Audio Volume.
- Initialize the video interface.
- Get the number of video channels supported by the camera.
- Register the call back function with the channel number as the user parameter to know from which the channel the video data is received.
- Gets and sets various channel paramters.
- Start the audio capture and video capture on all the all the channels for a duration in secs specified by the first argument to the application, if not specified it is default to 15 secs.
- Sets various channel paramters on the fly.
- The multi-channel video and audio data received in the call back functions are written into different files
- Stop the audio capture and video capture on all the all the channels.
- Free the resources allocated.

8.2. Alerts

The example `alerts.c` demonstrates multi-channel video and AAC audio capture along with configuring and receiving motion and audio alert statistics.

The programming flow of the application is as below.

- Initialize the audio interface.
- Set the Audio format and the sample rate.
- Register the audio call back function to receive audio data.
- Set the Audio Volume.
- Initialize the video interface.
- Get the number of video channels supported by the camera.
- Register the call back function with the channel number as the user parameter to know from which the channel the video data is received.
- Gets and sets various channel paramters.
- Initialize the Alert Subsystem.
- Enable the audio alerts with the audio alert call-back function with specific audio threshold.
- Enable the motion Alerts with the motion alert call-back function.
- Add the Rectangle on which the motion alert is required to be triggered.
- Start the audio capture and video capture on all the all the channels for a duration in secs specified by the first argument to the application, if not specified it is default to 15 secs.
- Sets various channel paramters on the fly.
- Enable and disable motion and audio alerts and get the current audio intensity during capture.
- The multi-channel video and audio data received in the call back functions are written into different files.
- Print the Motion and the Audio Alert statistics in the respective call-back functions.
- Stop the audio capture and video capture on all the all the channels.
- Free the resources allocated.