Developing with mxuvc

March 28, 2013

Table of Contents

1. Understanding mxuvc

mxuvc is a user-friendly API that enables developers to control and capture audio/video from a Geo camera.

Geo cameras are UVC/UAC compatible. Developers that have sufficient experience can directly communicate with the camera through the use of the V4L2 UVC Linux driver for video and ALSA for audio. However for both newcomers to UVC/UAC and experienced developers in that area, we do recommend using the mxuvc API because of its simplicity.

mxuvc relies on a video backend and an audio backend. There are two video backends and two audio backends available.

Video backends:

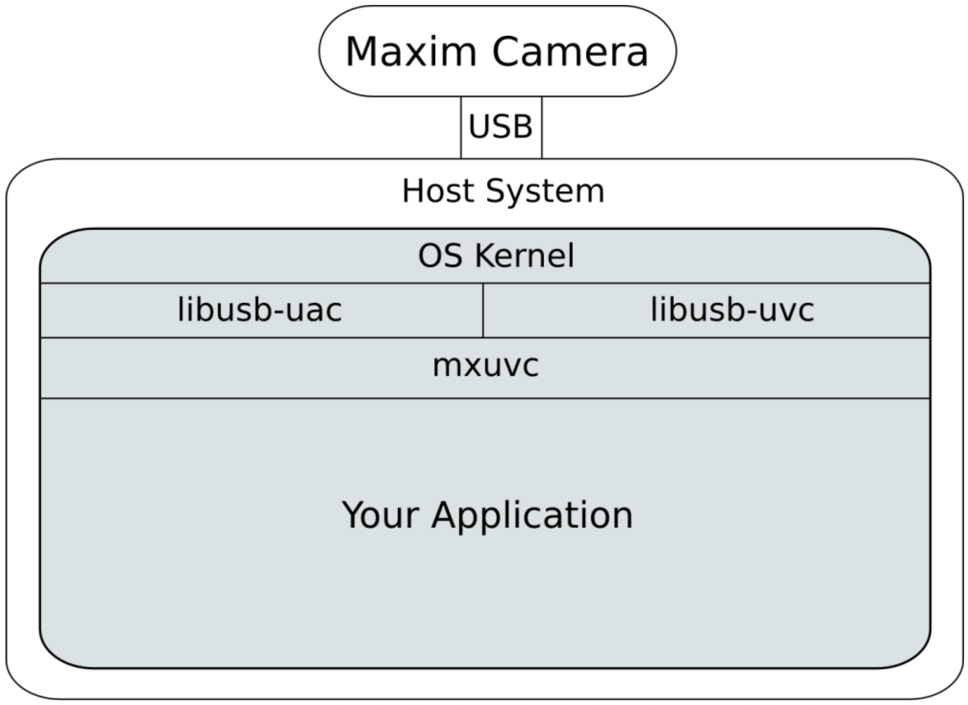
- `v4l2`: uses the linux kernel V4L2 UVC driver
- `libusb-uvc`: uses a libusb based user-space UVC driver

Audio backends:

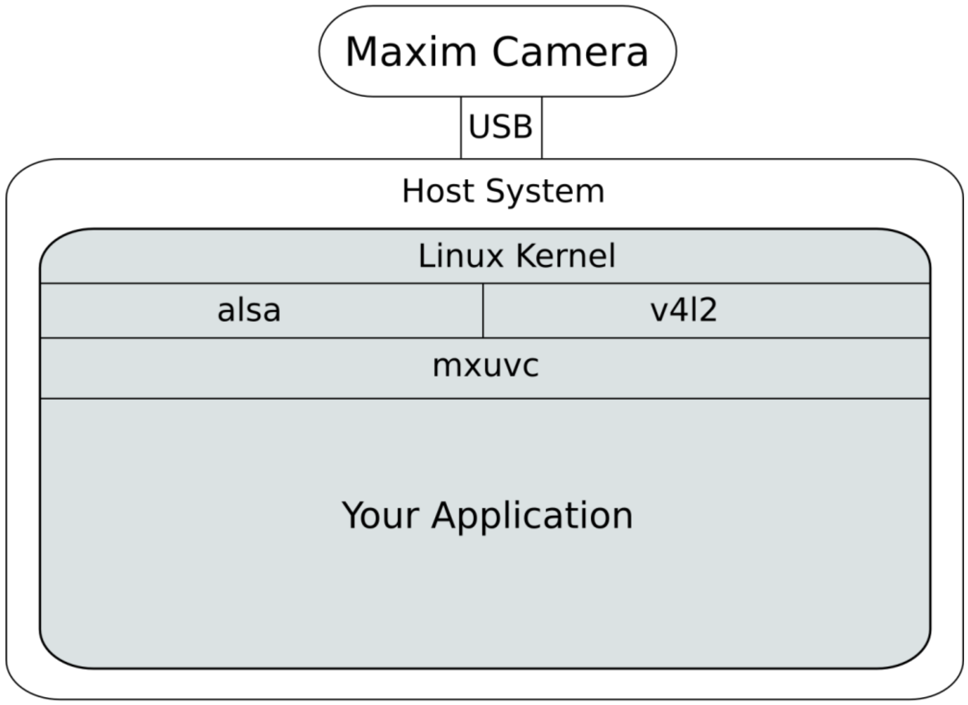
- `alsa`: uses the linux kernel ALSA UAC driver
- `libusb-uac`: uses a libusb based user-space UAC driver

Typically developers would use the `v4l2` backend for video and the `alsa` backend for audio since they rely on standard Linux kernel drivers. However in some rare cases, the use of a Linux kernel driver is not possible (for security concerns or because of the use of a non-Linux platform) and this is when the `libusb-uvc` and `libusb-uac` backends becomes necessary.
The audio and video backends are independent, therefore they can be mixed. For example if ALSA is available on the target platform but not V4L2, the alsa backend can be used for audio in conjunction to the `libusb-uvc` backend for video.

Whatever backend is chosen (build time choice), the API stays the same.



Using mxuvc with the libusb backends (libusb-uac and libusb-uvc)



Using mxuvc with the Linux kernel backends (alsa and v4l2)

2. 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:

```
# make AUDIO=<audio-backend> VIDEO=<video-backend>
```

`<audio-backend>` should be replaced by either `alsa` or `libusb-uac` depending on your choice.
`<video-backend>` should be replaced by either `v4l2` or `libusb-uvc` depending on your choice.
For example, to compile mxuvc with alsa and v4l2 as backends, type:

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

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.

3. Using mxuvc for video

This chapter gives a quick introduction on how to develop with mxuvc for video. Example applications are available in the examples/ directory and they should be the start point after reading this chapter.

The functions and types that are part of mxuvc are listed in `mxuvc.h`. This is the only header file to include in an application using this API. All the functions return a negative value in case of error.

Most functions take a *video channel* as first argument. The *video channel* can be either `CH_MAIN` or `CH_PREVIEW`.
`CH_MAIN` represents the first channel (also called main channel) of the camera: this is the channel whose video would be sent to the remote during a video conference call.
`CH_PREVIEW` represents the second channel (also called preview channel) of the camera: this is the channel whose video would be displayed locally during a video conference call.

The programming flow is as follow:

1	Initializing the video backend	`mxuvc_video_init()`
2	Registering a callback function	`mxuvc_video_register_cb()`
3	Choosing the video format	`mxuvc_video_set_format()`

4	Starting capturing video
	Checking whether the camera is working
	Stopping capturing video
		`mxuvc_video_start()`
		`mxuvc_video_alive()`
		`mxuvc_video_stop()`

5	Changing video parameters	
6	Freeing the resources	`mxuvc_video_deinit()`

1.1. Initializing the video backend

`mxuvc_video_init()` is used to initialized the video backend. This is the only function that differs between the `v4l2` and `libusb-uvc` backends.

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

Arguments, v4l2 backend:
- `backend` is a string representing the backend. It is "`v4l2`" in this case.
- `options` is a semi-colon separated list of options:

Option	Mandatory?	Description
`dev_offset`	no	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 and will default to 0 if omitted.
`dev_offset_secondary`	no	This is the offset for video node corresponding to preview channel. 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`	no	Number of memory mapped buffers for each device.

Here is an example:

```
      mxuvc_video_init("v4l2", "dev_offset=0");
```

or

```
      mxuvc_video_init("v4l2", "");
```

Arguments, libusb-uvc backend:

- `backend` is a string representing the backend. It is "`libusb-uvc`" in this case.
- `options` is a semi-colon separated list of options:

Option	Mandatory?	Description
`vid`	yes	This is the USB Vendor ID of the camera
`pid`	yes	This is the USB Product ID of the camera
`check_h264_continuity`	no	Ask the camera for a TS stream, demux it, check for continuity errors and then send the demuxed H264 elementary stream to the callback function. Print out debug messages whenever a continuity error is detected. This is mainly for debugging. 1 to enable, 0 to disable. Defaults to 0.
`packets_per_transfer`	no	Low level USB parameters. Defaults to 270. Change only in case of performance issues. Contact Geo for more information.

Option	Mandatory?	Description
`num_transfer`	no	Low level USB parameters. Defaults to 4. Change only in case of performance issues. Contact Geo for more information.

Here is an example:

```
      mxuvc_video_init("libusb-uvc","\
                   vid = 0xb6a;\
                   pid = 0x4d52;\
                   packets_per_transfer = 270;\
                   num_transfers = 4;\
                   check_h264_continuity=0");
```

or simply

```
      mxuvc_video_init("libusb-uvc","vid = 0xb6a; pid = 0x4d52");
```

1.2. Registering a callback function

The callback function is called every time a video frame is available. Only one function can be registered at a time. The processing done inside this function should be as minimal as possible (a few milliseconds) since it is blocking the video capture.

Declaration:

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

Arguments:

Argument	Description
`ch`	the video channel to register the callback function for.
`func`	the function to register as a callback
`user_data`	a pointer passed every time the callback function is called

A callback function should be declared as follow:

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

Argument	Description
buffer	the buffer containing the video frame
size	the size of the video frame
info	a structure containing information about the video frame like the format of the video the frame corresponds to or the timestamp.
user_data	the pointer given during `mxuvc_register_video_cb()`

1.3. Choosing the video format

The video format can be set using:

```
int mxuvc_video_set_format(video_channel_t ch,
                video_format_t fmt);
```

The video format, `fmt`, can be any of the following:

Video format	Channel support	Description
`VID_FORMAT_H264_RAW`	`CH_MAIN,` `CH_PREVIEW`	H264 elementary stream
`VID_FORMAT_H264_TS`	`CH_MAIN,` `CH_PREVIEW`	H264 encapsulated into a Transport Stream (TS)
`VID_FORMAT_MJPEG_RAW`	`CH_PREVIEW`	Raw MJPEG
`VID_FORMAT_YUY2_RAW`	`CH_PREVIEW`	YUY2 raw video
`VID_FORMAT_NV12_RAW`	`CH_PREVIEW`	NV12 raw video: not supported in MAX64380
`VID_FORMAT_H264_AAC_TS`	`CH_MAIN`	A Transport Stream containing H264 video and AAC audio (the audio is received through the video channel in this case)

1.4. Starting/Stopping capturing video

Once the video backend has been initialized and a callback function registered, the video capture can be started using:

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

And later stopped using:

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

The only argument is the video channel to start capturing from.

After the video has been started, the following function can periodically be called to check whether the camera is properly working:

```
int mxuvc_video_alive();
```

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*

In case of error, the video backend should be uninitialized using `mxuvc_video_deinit` (see the section “Freeing the ressources”).

1.5. Changing video parameters

At any moment after the backend has been initialized, the video parameters can be changed. The functions to achieve that purpose make the most part of the API. The most important ones are:

Description	Value range	Function
Change resolution	From 160x120 to 1920x1080	`mxuvc_video_set_resolution()`
Change the frame rate	1 to 30	`mxuvc_video_set_framerate()`
Change the video bit rate (in bits)	0 to 8000000	`mxuvc_video_set_bitrate()`
Change the GOP size (for H264 formats)	0 to 2147483647 0 is infinite	`mxuvc_video_set_goplen()`
Force an I Frame (for H264 formats)	1	`mxuvc_video_force_iframe()`
Change the profile (for H264 formats)	PROFILE_BASELINE PROFILE_MAIN PROFILE_HIGH	`mxuvc_video_set_profile()`
Set the maximum size of a NAL unit (for H264 formats)	0 to 2000	`mxuvc_video_set_maxnal()`
Flip the video vertically	FLIP_OFF or FLIP_ON	`mxuvc_video_set_flip_vertical()`
Flip the video horizontally	FLIP_OFF or FLIP_ON	`mxuvc_video_set_flip_horizontal()`
Set the maximum size of a frame	0 to 64000 0 is not maximum	`mxuvc_video_set_max_framesize()`
Zoom into the video	0 to 100	`mxuvc_video_set_zoom()`
Pan and tilt through the video	pan: -648000 to 648000 tilt: -648000 to 648000	`mxuvc_video_set_pantilt()`

The other functions are used for debugging and tuning only since the parameters they control are dynamically controlled by the camera firmware and are supposed to be correct.

Description	Value range	Function
Change the brightness	-255 to 255	`mxuvc_video_set_brightness()`
Change the contrast	0 to 200	`mxuvc_video_set_contrast()`
Change the hue	-18000 to 18000	`mxuvc_video_set_hue()`

Description	Value range	Function
Change the saturation	0 to 200	`mxuvc_video_set_saturation()`
Change the gain	1 to 100	`mxuvc_video_set_gain()`
Change the gamma	100 to 300	`mxuvc_video_set_gamma()`
Change the sharpness	0 to 100	`mxuvc_video_set_sharpness()`
Manually adjust the Wide Dynamic Range control's intensity.	WDR_AUTO or WDR_MANUAL (0 to 255)	`mxuvc_video_set_wdr()`
Controls the maximum sensor analog gain in auto exposure algorithm.	0 to 15	`mxuvc_video_set_max_analog_gain()`
Disables/enables histogram equalization, which creates more contrast to the image.	0 to disable. 1 to enable	`mxuvc_video_set_histogram_eq()`
Set different strengths of the sharpening filter.	0, 1 or 2 2 is the strongest	`mxuvc_video_set_sharpen_filter()`
Controls the auto exposure algorithm to adjust the sensor analog gain and exposure based on different lighting conditions.	0 to 256	`mxuvc_video_set_gain_multiplier()`

1.6. Freeing the resources

When the video backend is not needed anymore, all the memory allocated by it can be freed with:

```
int mxuvc_video_deinit();
```

This function automatically `mxuvc_video_stop()` if the video has not been stopped yet: it is therefore not necessary to call `mxuvc_video_stop()` before calling `mxuvc_video_deinit()`.

4. Using mxuvc for audio

This chapter gives a quick introduction on how to develop with mxuvc for audio. Example applications are available in the examples/ directory and they should be the start point after reading this chapter.

The functions and types that are part of mxuvc are listed in `mxuvc.h`. This is the only header file to include in an application using this API. All the functions return a negative value in case of error.

The programming flow is as follow:

1	Initializing the audio backend	`mxuvc_audio_init()`
2	Registering a callback function	`mxuvc_audio_register_cb()`
3	Choosing audio format	`mxuvc_audio_set_format()`

4	Starting capturing audio
	Checking whether the camera is working
	Stopping capturing audio
		`mxuvc_audio_start()`
		`mxuvc_audio_alive()`
		`mxuvc_audio_stop()`

5	Changing audio parameters	
6	Freeing the resources	`mxuvc_audio_deinit()`

1.1. Initializing the audio backend

`mxuvc_audio_init()` is used to initialized the audio backend. This is the only function that differs between the alsa and `libusb-uac` backends.

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

Arguments, alsa backend:

- `backend` is a string representing the backend. It is "`alsa`" in this case.
- `options` is a semi-colon separated list of options.

Option	Mandatory?	Description
`device`	yes	ALSA name of the camera. Typically MAX64380.
`audio_sampling_rate`	no	Audio sampling rate to initialize the camera with. Can be changed later with `mxuvc_audio_set_samplerate()`. Defaults to 24000 (24 kHz).

Option	Mandatory?	Description
`audio_duration_ms`	no	Number of milliseconds of audio to get before invoking the callback function. Defaults to 10.
`audio_channel_count`	no	Number of channel to get the audio from. Defaults to 2.

Here is an example:

```
     mxuvc_audio_init("alsa","\
                    device = MAX64380;\
                    audio_sampling_rate = 24000; \
                    audio_channel_count = 2; \
                    audio_duration_ms = 10");
or simply
     mxuvc_audio_init("alsa", “device = MAX64380”);
```

Arguments, libusb-uac backend:

- `backend` is a string representing the backend. It is “`libusb-uac`” in this case.
- `options` is a semi-colon separated list of options:

Option	Mandatory?	Description
`vid`	yes	This is the USB Vendor ID of the camera
`pid`	yes	This is the USB Product ID of the camera
`audio_sampling_rate`	no	Audio sampling rate to initialize the camera with. Can be changed later with `mxuvc_audio_set_samplerate()`. Defaults to 24000 (24 kHz).
`audio_duration_ms`	no	Number of milliseconds of audio to get before invoking the callback function. Defaults to 10. **Must** be 10 for the `libusb-uac` backend.
`packets_per_transfer`	no	Low level USB parameters. Defaults to 5. Change only in case of performance issues. Contact Geo for more information.
`num_transfer`	no	Low level USB parameters. Defaults to 50. Change only in case of performance issues. Contact Geo for more information.

Here is an example:

```
      mxuvc_audio_init("libusb-uac", "\
                           vid = 0xb6a;\
                           pid = 0x4d52;\
                           packets_per_transfer = 5;\
                           num_transfers = 50;\
                           audio_duration_ms = 10;\
                           audio_sampling_rate = 24000");
```

or simply

```
      mxuvc_audio_init("libusb-uvc","vid = 0xb6a; pid = 0x4d52");
```

1.2. Registering a callback function

The callback function is called every time an audio frame is available. Only one function can be registered at a time. The processing done inside this function should be as minimal as possible (a few milliseconds) since it is blocking the video capture.

Declaration:

```
int mxuvc_audio_register_cb(mxuvc_cb_t func, void *user_data);
```

Arguments:

Argument	Description
`func`	the function to register as a callback
`user_data`	a pointer passed every time the callback function is called

A callback function should be declared as follow:

```
void example_cb(unsigned char *buffer, unsigned int size,
                int format, uint32_t ts, void *user_data);
```

Argument	Description
`buffer`	the buffer containing the audio data
`size`	the size of the audio data
`format`	the format of the audio the frame corresponds to
`ts`	the timestamp
`user_data`	the pointer given during `mxuvc_register_audio_cb()`

1.3. Choosing the audio format

The audio format can be set using:

```
int mxuvc_set_audio_format(audio_format_t fmt);
```

The audio format, `fmt`, can be any of the following:

Audio format	Description
`AUD_FORMAT_PCM_RAW`	Raw PCM audio data
`AUD_FORMAT_AAC_RAW`	AAC audio data: not supported in MAX64380.

1.4. Starting/Stopping capturing audio

Once the audio backend has been initialized and a callback function registered, the audio capture can be started using:

```
int mxuvc_audio_start();
```

And later stopped using:

```
int mxuvc_audio_stop();
```

After the audio has been started, the following function can periodically be called to check whether the camera is properly working:

```
int mxuvc_audio_alive();
```

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*

In case of error, the audio backend should be uninitialized using `mxuvc_audio_deinit()` (see the section “Freeing the ressources”).

1.5. Changing audio parameters

At any moment after the backend has been initialized, the audio parameters can be changed.

Description	Value range	Function
Change the sampling rate	8000, 16000 or 24000	`mxuvc_audio_set_samplerate()`
Change the volume	0 to 100	`mxuvc_audio_set_volume()`
Mute/Unmute	0 to unmute, 1 to mute	`mxuvc_audio_set_mic_mute()`

1.6. Freeing the resources

When the audio backend is not needed anymore, all the memory allocated by it can be freed with:

```
    int mxuvc_audio_deinit();
```

This function automatically `mxuvc_audio_stop()` if the audio has not been stopped yet: it is therefore not necessary to call `mxuvc_audio_stop()` before calling `mxuvc_audio_deinit()`.