Video Data Types

video_format_t

The `video_format_t` data type defined by

```
typedef enum {
    VIDEO_FORMAT_4_3,     /* Select 4:3 format */
    VIDEO_FORMAT_16_9,    /* Select 16:9 format. */
    VIDEO_FORMAT_221_1    /* 2.21:1 */
} video_format_t;
```

is used in the VIDEO_SET_FORMAT function (??) to tell the driver which aspect ratio the output hardware (e.g. TV) has. It is also used in the data structures video_status (??) returned by VIDEO_GET_STATUS (??) and video_event (??) returned by VIDEO_GET_EVENT (??) which report about the display format of the current video stream.

video_displayformat_t

In case the display format of the video stream and of the display hardware differ the application has to specify how to handle the cropping of the picture. This can be done using the VIDEO_SET_DISPLAY_FORMAT call (??) which accepts

```
typedef enum {
    VIDEO_PAN_SCAN,       /* use pan and scan format */
    VIDEO_LETTER_BOX,     /* use letterbox format */
    VIDEO_CENTER_CUT_OUT  /* use center cut out format */
} video_displayformat_t;
```

as argument.

video_stream_source_t

The video stream source is set through the VIDEO_SELECT_SOURCE call and can take the following values, depending on whether we are replaying from an internal (demuxer) or external (user write) source.

```
typedef enum {
    VIDEO_SOURCE_DEMUX, /* Select the demux as the main source */
    VIDEO_SOURCE_MEMORY /* If this source is selected, the stream
                   comes from the user through the write
                   system call */
} video_stream_source_t;
```

VIDEO_SOURCE_DEMUX selects the demultiplexer (fed either by the frontend or the DVR device) as the source of the video stream. If VIDEO_SOURCE_MEMORY is selected the stream comes from the application through the **write()** system call.

video_play_state_t

The following values can be returned by the VIDEO_GET_STATUS call representing the state of video playback.

```
typedef enum {
    VIDEO_STOPPED, /* Video is stopped */
    VIDEO_PLAYING, /* Video is currently playing */
    VIDEO_FREEZED  /* Video is freezed */
} video_play_state_t;
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\drivers\staging\media\av7110\(linux-master)(drivers)(staging)(media)(av7110)video_types.rst, line 98)

Unknown directive type "c:type".

```
.. c:type:: video_command
```

struct video_command

The structure must be zeroed before use by the application This ensures it can be extended safely in the future.

```
struct video_command {
    __u32 cmd;
    __u32 flags;
```

```
    union {
        struct {
            __u64 pts;
        } stop;

        struct {
            /* 0 or 1000 specifies normal speed,
               1 specifies forward single stepping,
               -1 specifies backward single stepping,
               >>1: playback at speed/1000 of the normal speed,
               <-1: reverse playback at (-speed/1000) of the normal speed. */
            __s32 speed;
            __u32 format;
        } play;

        struct {
            __u32 data[16];
        } raw;
    };
};
```

video_size_t

```
typedef struct {
    int w;
    int h;
    video_format_t aspect_ratio;
} video_size_t;
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\drivers\staging\media\av7110\(linux-master)(drivers)(staging)(media)(av7110)video_types.rst, line 149)

Unknown directive type "c:type".

```
.. c:type:: video_event
```

struct video_event

The following is the structure of a video event as it is returned by the VIDEO_GET_EVENT call.

```
struct video_event {
     __s32 type;
#define VIDEO_EVENT_SIZE_CHANGED    1
#define VIDEO_EVENT_FRAME_RATE_CHANGED  2
#define VIDEO_EVENT_DECODER_STOPPED     3
#define VIDEO_EVENT_VSYNC       4
    long timestamp;
    union {
        video_size_t size;
        unsigned int frame_rate;    /* in frames per 1000sec */
        unsigned char vsync_field;  /* unknown/odd/even/progressive */
    } u;
};
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\drivers\staging\media\av7110\(linux-master)(drivers)(staging)(media)(av7110)video_types.rst, line 175)

Unknown directive type "c:type".

```
.. c:type:: video_status
```

struct video_status

The VIDEO_GET_STATUS call returns the following structure informing about various states of the playback operation.

```
struct video_status {
    int                   video_blank;   /* blank video on freeze? */
    video_play_state_t    play_state;    /* current state of playback */
    video_stream_source_t stream_source; /* current source (demux/memory) */
    video_format_t        video_format;  /* current aspect ratio of stream */
    video_displayformat_t display_format;/* selected cropping mode */
```

```
};
```

If video_blank is set video will be blanked out if the channel is changed or if playback is stopped. Otherwise, the last picture will be displayed. play_state indicates if the video is currently frozen, stopped, or being played back. The stream_source corresponds to the selected source for the video stream. It can come either from the demultiplexer or from memory. The video_format indicates the aspect ratio (one of 4:3 or 16:9) of the currently played video stream. Finally, display_format corresponds to the selected cropping mode in case the source video format is not the same as the format of the output device.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\drivers\staging\media\av7110\(linux-master)(drivers)(staging)(media)(av7110)video_types.rst, line 206)

Unknown directive type "c:type".

```
.. c:type:: video_still_picture
```

struct video_still_picture

An I-frame displayed via the VIDEO_STILLPICTURE call is passed on within the following structure.

```
/* pointer to and size of a single iframe in memory */
struct video_still_picture {
    char *iFrame;        /* pointer to a single iframe in memory */
    int32_t size;
};
```

video capabilities

A call to VIDEO_GET_CAPABILITIES returns an unsigned integer with the following bits set according to the hardwares capabilities.

```
/* bit definitions for capabilities: */
/* can the hardware decode MPEG1 and/or MPEG2? */
#define VIDEO_CAP_MPEG1   1
#define VIDEO_CAP_MPEG2   2
/* can you send a system and/or program stream to video device?
   (you still have to open the video and the audio device but only
    send the stream to the video device) */
#define VIDEO_CAP_SYS     4
#define VIDEO_CAP_PROG    8
/* can the driver also handle SPU, NAVI and CSS encoded data?
   (CSS API is not present yet) */
#define VIDEO_CAP_SPU    16
#define VIDEO_CAP_NAVI   32
#define VIDEO_CAP_CSS    64
```