Linux Media Subsystem Documentation

Release

The kernel development community

Feb 08, 2017

CONTENTS

Contents:

CHAPTER

ONE

LINUX MEDIA INFRASTRUCTURE USERSPACE API

Table of Contents

1.1 Introduction

This document covers the Linux Kernel to Userspace API’s used by video and radio streaming devices, including video cameras, analog and digital TV receiver cards, AM/FM receiver cards, Software Defined Radio (SDR), streaming capture and output devices, codec devices and remote controllers.

A typical media device hardware is shown at *Typical Media Device* .

Fig. 1.1: Typical Media Device

The media infrastructure API was designed to control such devices. It is divided into five parts.

1. The *first part* covers radio, video capture and output, cameras, analog TV devices and codecs.
2. The *second part* covers the API used for digital TV and Internet reception via one of the several digital tv standards. While it is called as DVB API, in fact it covers several different video standards including DVB-T/T2, DVB-S/S2, DVB-C, ATSC, ISDB-T, ISDB-S, DTMB, etc. The complete list of supported standards can be found at *fe_delivery_system*.
3. The *third part* covers the Remote Controller API.
4. The *fourth part* covers the Media Controller API.
5. The *fifth part* covers the CEC (Consumer Electronics Control) API.

It should also be noted that a media device may also have audio components, like mixers, PCM capture, PCM playback, etc, which are controlled via ALSA API. For additional information and for the latest development code, see: https://linuxtv.org. For discussing improvements, reporting troubles, sending new drivers, etc, please mail to: Linux Media Mailing List (LMML).

1.2 Part I - Video for Linux API

This part describes the Video for Linux API version 2 (V4L2 API) specification.

Revision 4.5

Table of Contents

1.2.1 Common API Elements

Programming a V4L2 device consists of these steps:

- Opening the device
- Changing device properties, selecting a video and audio input, video standard, picture brightness a. o.
- Negotiating a data format
- Negotiating an input/output method
- The actual input/output loop
- Closing the device

In practice most steps are optional and can be executed out of order. It depends on the V4L2 device type, you can read about the details in *Interfaces* . In this chapter we will discuss the basic concepts applicable to all devices.

Opening and Closing Devices

Device Naming

V4L2 drivers are implemented as kernel modules, loaded manually by the system administrator or automatically when a device is first discovered. The driver modules plug into the "videodev" kernel module. It provides helper functions and a common application interface specified in this document.

Each driver thus loaded registers one or more device nodes with major number 81 and a minor number between 0 and 255. Minor numbers are allocated dynamically unless the kernel is compiled with the kernel option CONFIG_VIDEO_FIXED_MINOR_RANGES. In that case minor numbers are allocated in ranges depending on the device node type (video, radio, etc.).

Many drivers support "video_nr", "radio_nr" or "vbi_nr" module options to select specific video/radio/vbi node numbers. This allows the user to request that the device node is named e.g. /dev/video5 instead of leaving it to chance. When the driver supports multiple devices of the same type more than one device node number can be assigned, separated by commas:

```
# modprobe mydriver video_nr=0,1 radio_nr=0,1
```

In `/etc/modules.conf` this may be written as:

```
options mydriver video_nr=0,1 radio_nr=0,1
```

When no device node number is given as module option the driver supplies a default.

Normally udev will create the device nodes in /dev automatically for you. If udev is not installed, then you need to enable the CONFIG_VIDEO_FIXED_MINOR_RANGES kernel option in order to be able to correctly relate a minor number to a device node number. I.e., you need to be certain that minor number 5 maps to device node name video5. With this kernel option different device types have different minor number ranges. These ranges are listed in *Interfaces* .

The creation of character special files (with mknod) is a privileged operation and devices cannot be opened by major and minor number. That means applications cannot *reliable* scan for loaded or installed drivers. The user must enter a device name, or the application can try the conventional device names.

Related Devices

Devices can support several functions. For example video capturing, VBI capturing and radio support.

The V4L2 API creates different nodes for each of these functions.

The V4L2 API was designed with the idea that one device node could support all functions. However, in practice this never worked: this 'feature' was never used by applications and many drivers did not support it and if they did it was certainly never tested. In addition, switching a device node between different functions only works when using the streaming I/O API, not with the *read()* / *write()* API.

Today each device node supports just one function.

Besides video input or output the hardware may also support audio sampling or playback. If so, these functions are implemented as ALSA PCM devices with optional ALSA audio mixer devices.

One problem with all these devices is that the V4L2 API makes no provisions to find these related devices. Some really complex devices use the Media Controller (see *Part IV - Media Controller API*) which can be used for this purpose. But most drivers do not use it, and while some code exists that uses sysfs to discover related devices (see libmedia_dev in the v4l-utils git repository), there is no library yet that can provide a single API towards both Media Controller-based devices and devices that do not use the Media Controller. If you want to work on this please write to the linux-media mailing list: https://linuxtv.org/lists.php.

Multiple Opens

V4L2 devices can be opened more than once. [1] When this is supported by the driver, users can for example start a "panel" application to change controls like brightness or audio volume, while another application captures video and audio. In other words, panel applications are comparable to an ALSA audio mixer application. Just opening a V4L2 device should not change the state of the device. [2]

Once an application has allocated the memory buffers needed for streaming data (by calling the *ioctl VIDIOC_REQBUFS* or *ioctl VIDIOC_CREATE_BUFS* ioctls, or implicitly by calling the *read()* or *write()* functions) that application (filehandle) becomes the owner of the device. It is no longer allowed to make changes that would affect the buffer sizes (e.g. by calling the *VIDIOC_S_FMT* ioctl) and other applications are no longer allowed to allocate buffers or start or stop streaming. The EBUSY error code will be returned instead.

Merely opening a V4L2 device does not grant exclusive access. [3] Initiating data exchange however assigns the right to read or write the requested type of data, and to change related properties, to this file descriptor. Applications can request additional access privileges using the priority mechanism described in *Application Priority* .

Shared Data Streams

V4L2 drivers should not support multiple applications reading or writing the same data stream on a device by copying buffers, time multiplexing or similar means. This is better handled by a proxy application in user space.

Functions

To open and close V4L2 devices applications use the *open()* and *close()* function, respectively. Devices are programmed using the *ioctl()* function as explained in the following sections.

Querying Capabilities

Because V4L2 covers a wide variety of devices not all aspects of the API are equally applicable to all types of devices. Furthermore devices of the same type have different capabilities and this specification permits the omission of a few complicated and less important parts of the API.

[1] There are still some old and obscure drivers that have not been updated to allow for multiple opens. This implies that for such drivers *open()* can return an EBUSY error code when the device is already in use.

[2] Unfortunately, opening a radio device often switches the state of the device to radio mode in many drivers. This behavior should be fixed eventually as it violates the V4L2 specification.

[3] Drivers could recognize the O_EXCL open flag. Presently this is not required, so applications cannot know if it really works.

The *ioctl VIDIOC_QUERYCAP* ioctl is available to check if the kernel device is compatible with this specification, and to query the *functions* and *I/O methods* supported by the device.

Starting with kernel version 3.1, *ioctl VIDIOC_QUERYCAP* will return the V4L2 API version used by the driver, with generally matches the Kernel version. There's no need of using *ioctl VIDIOC_QUERYCAP* to check if a specific ioctl is supported, the V4L2 core now returns EN0TTY if a driver doesn't provide support for an ioctl.

Other features can be queried by calling the respective ioctl, for example *ioctl VIDIOC_ENUMINPUT* to learn about the number, types and names of video connectors on the device. Although abstraction is a major objective of this API, the *ioctl VIDIOC_QUERYCAP* ioctl also allows driver specific applications to reliably identify the driver.

All V4L2 drivers must support *ioctl VIDIOC_QUERYCAP* . Applications should always call this ioctl after opening the device.

Application Priority

When multiple applications share a device it may be desirable to assign them different priorities. Contrary to the traditional "rm -rf /" school of thought a video recording application could for example block other applications from changing video controls or switching the current TV channel. Another objective is to permit low priority applications working in background, which can be preempted by user controlled applications and automatically regain control of the device at a later time.

Since these features cannot be implemented entirely in user space V4L2 defines the *VIDIOC_G_PRIORITY* and *VIDIOC_S_PRIORITY* ioctls to request and query the access priority associate with a file descriptor. Opening a device assigns a medium priority, compatible with earlier versions of V4L2 and drivers not supporting these ioctls. Applications requiring a different priority will usually call *VIDIOC_S_PRIORITY* after verifying the device with the *ioctl VIDIOC_QUERYCAP* ioctl.

Ioctls changing driver properties, such as *VIDIOC_S_INPUT* , return an EBUSY error code after another application obtained higher priority.

Video Inputs and Outputs

Video inputs and outputs are physical connectors of a device. These can be for example RF connectors (antenna/cable), CVBS a.k.a. Composite Video, S-Video or RGB connectors. Video and VBI capture devices have inputs. Video and VBI output devices have outputs, at least one each. Radio devices have no video inputs or outputs.

To learn about the number and attributes of the available inputs and outputs applications can enumerate them with the *ioctl VIDIOC_ENUMINPUT* and *ioctl VIDIOC_ENUMOUTPUT* ioctl, respectively. The struct *v4l2_input* returned by the *ioctl VIDIOC_ENUMINPUT* ioctl also contains signal :status information applicable when the current video input is queried.

The *VIDIOC_G_INPUT* and *VIDIOC_G_OUTPUT* ioctls return the index of the current video input or output. To select a different input or output applications call the *VIDIOC_S_INPUT* and *VIDIOC_S_OUTPUT* ioctls. Drivers must implement all the input ioctls when the device has one or more inputs, all the output ioctls when the device has one or more outputs.

Example: Information about the current video input

```
struct v4l2_input input;
int index;

if (-1 == ioctl(fd, VIDIOC_G_INPUT, &index)) {
    perror("VIDIOC_G_INPUT");
    exit(EXIT_FAILURE);
}
```

```
memset(&input, 0, sizeof(input));
input.index = index;

if (-1 == ioctl(fd, VIDIOC_ENUMINPUT, &input)) {
    perror("VIDIOC_ENUMINPUT");
    exit(EXIT_FAILURE);
}

printf("Current input: %s\\n", input.name);
```

Example: Switching to the first video input

```
int index;

index = 0;

if (-1 == ioctl(fd, VIDIOC_S_INPUT, &index)) {
    perror("VIDIOC_S_INPUT");
    exit(EXIT_FAILURE);
}
```

Audio Inputs and Outputs

Audio inputs and outputs are physical connectors of a device. Video capture devices have inputs, output devices have outputs, zero or more each. Radio devices have no audio inputs or outputs. They have exactly one tuner which in fact *is* an audio source, but this API associates tuners with video inputs or outputs only, and radio devices have none of these. [1] A connector on a TV card to loop back the received audio signal to a sound card is not considered an audio output.

Audio and video inputs and outputs are associated. Selecting a video source also selects an audio source. This is most evident when the video and audio source is a tuner. Further audio connectors can combine with more than one video input or output. Assumed two composite video inputs and two audio inputs exist, there may be up to four valid combinations. The relation of video and audio connectors is defined in the `audioset` field of the respective struct *v4l2_input* or struct *v4l2_output*, where each bit represents the index number, starting at zero, of one audio input or output.

To learn about the number and attributes of the available inputs and outputs applications can enumerate them with the *ioctl VIDIOC_ENUMAUDIO* and *VIDIOC_ENUMAUDOUT* ioctl, respectively. The struct *v4l2_audio* returned by the *ioctl VIDIOC_ENUMAUDIO* ioctl also contains signal :status information applicable when the current audio input is queried.

The *VIDIOC_G_AUDIO* and *VIDIOC_G_AUDOUT* ioctls report the current audio input and output, respectively.

Note:

Note that, unlike VIDIOC_G_INPUT *and* VIDIOC_G_OUTPUT *these ioctls return a structure as* ioctl VIDIOC_ENUMAUDIO *and* VIDIOC_ENUMAUDOUT *do, not just an index.*

To select an audio input and change its properties applications call the *VIDIOC_S_AUDIO* ioctl. To select an audio output (which presently has no changeable properties) applications call the *VIDIOC_S_AUDOUT* ioctl.

[1] Actually struct *v4l2_audio* ought to have a `tuner` field like struct *v4l2_input*, not only making the API more consistent but also permitting radio devices with multiple tuners.

Drivers must implement all audio input ioctls when the device has multiple selectable audio inputs, all audio output ioctls when the device has multiple selectable audio outputs. When the device has any audio inputs or outputs the driver must set the V4L2_CAP_AUDIO flag in the struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl.

Example: Information about the current audio input

```
struct v4l2_audio audio;

memset(&audio, 0, sizeof(audio));

if (-1 == ioctl(fd, VIDIOC_G_AUDIO, &audio)) {
    perror("VIDIOC_G_AUDIO");
    exit(EXIT_FAILURE);
}

printf("Current input: %s\\n", audio.name);
```

Example: Switching to the first audio input

```
struct v4l2_audio audio;

memset(&audio, 0, sizeof(audio)); /* clear audio.mode, audio.reserved */

audio.index = 0;

if (-1 == ioctl(fd, VIDIOC_S_AUDIO, &audio)) {
    perror("VIDIOC_S_AUDIO");
    exit(EXIT_FAILURE);
}
```

Tuners and Modulators

Tuners

Video input devices can have one or more tuners demodulating a RF signal. Each tuner is associated with one or more video inputs, depending on the number of RF connectors on the tuner. The type field of the respective struct *v4l2_input* returned by the *ioctl VIDIOC_ENUMINPUT* ioctl is set to V4L2_INPUT_TYPE_TUNER and its tuner field contains the index number of the tuner.

Radio input devices have exactly one tuner with index zero, no video inputs.

To query and change tuner properties applications use the *VIDIOC_G_TUNER* and *VIDIOC_S_TUNER* ioctls, respectively. The struct *v4l2_tuner* returned by *VIDIOC_G_TUNER* also contains signal status information applicable when the tuner of the current video or radio input is queried.

Note:

VIDIOC_S_TUNER *does not switch the current tuner, when there is more than one at all. The tuner is solely determined by the current video input. Drivers must support both ioctls and set the V4L2_CAP_TUNER flag in the struct* v4l2_capability *returned by the* ioctl VIDIOC_QUERYCAP *ioctl when the device has one or more tuners.*

Modulators

Video output devices can have one or more modulators, uh, modulating a video signal for radiation or connection to the antenna input of a TV set or video recorder. Each modulator is associated with one or more video outputs, depending on the number of RF connectors on the modulator. The `type` field of the respective struct *v4l2_output* returned by the *ioctl VIDIOC_ENUMOUTPUT* ioctl is set to `V4L2_OUTPUT_TYPE_MODULATOR` and its `modulator` field contains the index number of the modulator.

Radio output devices have exactly one modulator with index zero, no video outputs.

A video or radio device cannot support both a tuner and a modulator. Two separate device nodes will have to be used for such hardware, one that supports the tuner functionality and one that supports the modulator functionality. The reason is a limitation with the *VIDIOC_S_FREQUENCY* ioctl where you cannot specify whether the frequency is for a tuner or a modulator.

To query and change modulator properties applications use the *VIDIOC_G_MODULATOR* and *VIDIOC_S_MODULATOR* ioctl. Note that *VIDIOC_S_MODULATOR* does not switch the current modulator, when there is more than one at all. The modulator is solely determined by the current video output. Drivers must support both ioctls and set the `V4L2_CAP_MODULATOR` flag in the struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl when the device has one or more modulators.

Radio Frequency

To get and set the tuner or modulator radio frequency applications use the *VIDIOC_G_FREQUENCY* and *VIDIOC_S_FREQUENCY* ioctl which both take a pointer to a struct *v4l2_frequency*. These ioctls are used for TV and radio devices alike. Drivers must support both ioctls when the tuner or modulator ioctls are supported, or when the device is a radio device.

Video Standards

Video devices typically support one or more different video standards or variations of standards. Each video input and output may support another set of standards. This set is reported by the `std` field of struct *v4l2_input* and struct *v4l2_output* returned by the *ioctl VIDIOC_ENUMINPUT* and *ioctl VIDIOC_ENUMOUTPUT* ioctls, respectively.

V4L2 defines one bit for each analog video standard currently in use worldwide, and sets aside bits for driver defined standards, e. g. hybrid standards to watch NTSC video tapes on PAL TVs and vice versa. Applications can use the predefined bits to select a particular standard, although presenting the user a menu of supported standards is preferred. To enumerate and query the attributes of the supported standards applications use the *ioctl VIDIOC_ENUMSTD* ioctl.

Many of the defined standards are actually just variations of a few major standards. The hardware may in fact not distinguish between them, or do so internal and switch automatically. Therefore enumerated standards also contain sets of one or more standard bits.

Assume a hypothetic tuner capable of demodulating B/PAL, G/PAL and I/PAL signals. The first enumerated standard is a set of B and G/PAL, switched automatically depending on the selected radio frequency in UHF or VHF band. Enumeration gives a “PAL-B/G” or “PAL-I” choice. Similar a Composite input may collapse standards, enumerating “PAL-B/G/H/I”, “NTSC-M” and “SECAM-D/K”. [1]

To query and select the standard used by the current video input or output applications call the *VIDIOC_G_STD* and *VIDIOC_S_STD* ioctl, respectively. The *received* standard can be sensed with the *ioctl VIDIOC_QUERYSTD* ioctl.

[1] Some users are already confused by technical terms PAL, NTSC and SECAM. There is no point asking them to distinguish between B, G, D, or K when the software or hardware can do that automatically.

Note:

The parameter of all these ioctls is a pointer to a v4l2_std_id *type (a standard set),* not *an index into the standard enumeration. Drivers must implement all video standard ioctls when the device has one or more video inputs or outputs.*

Special rules apply to devices such as USB cameras where the notion of video standards makes little sense. More generally for any capture or output device which is:

- incapable of capturing fields or frames at the nominal rate of the video standard, or
- that does not support the video standard formats at all.

Here the driver shall set the `std` field of struct *v4l2_input* and struct *v4l2_output* to zero and the *VIDIOC_G_STD* , *VIDIOC_S_STD* , *ioctl VIDIOC_QUERYSTD* and *ioctl VIDIOC_ENUMSTD* ioctls shall return the `ENOTTY` error code or the `EINVAL` error code.

Applications can make use of the *Input capabilities* and *Output capabilities* flags to determine whether the video standard ioctls can be used with the given input or output.

Example: Information about the current video standard

```
v4l2_std_id std_id;
struct v4l2_standard standard;

if (-1 == ioctl(fd, VIDIOC_G_STD, &std_id)) {
    /* Note when VIDIOC_ENUMSTD always returns ENOTTY this
       is no video device or it falls under the USB exception,
       and VIDIOC_G_STD returning ENOTTY is no error. */

    perror("VIDIOC_G_STD");
    exit(EXIT_FAILURE);
}

memset(&standard, 0, sizeof(standard));
standard.index = 0;

while (0 == ioctl(fd, VIDIOC_ENUMSTD, &standard)) {
    if (standard.id & std_id) {
            printf("Current video standard: %s\\n", standard.name);
            exit(EXIT_SUCCESS);
    }

    standard.index++;
}

/* EINVAL indicates the end of the enumeration, which cannot be
   empty unless this device falls under the USB exception. */

if (errno == EINVAL || standard.index == 0) {
    perror("VIDIOC_ENUMSTD");
    exit(EXIT_FAILURE);
}
```

Example: Listing the video standards supported by the current input

```
struct v4l2_input input;
struct v4l2_standard standard;
```

```
memset(&input, 0, sizeof(input));

if (-1 == ioctl(fd, VIDIOC_G_INPUT, &input.index)) {
    perror("VIDIOC_G_INPUT");
    exit(EXIT_FAILURE);
}

if (-1 == ioctl(fd, VIDIOC_ENUMINPUT, &input)) {
    perror("VIDIOC_ENUM_INPUT");
    exit(EXIT_FAILURE);
}

printf("Current input %s supports:\\n", input.name);

memset(&standard, 0, sizeof(standard));
standard.index = 0;

while (0 == ioctl(fd, VIDIOC_ENUMSTD, &standard)) {
    if (standard.id & input.std)
        printf("%s\\n", standard.name);

    standard.index++;
}

/* EINVAL indicates the end of the enumeration, which cannot be
   empty unless this device falls under the USB exception. */

if (errno != EINVAL || standard.index == 0) {
    perror("VIDIOC_ENUMSTD");
    exit(EXIT_FAILURE);
}
```

Example: Selecting a new video standard

```
struct v4l2_input input;
v4l2_std_id std_id;

memset(&input, 0, sizeof(input));

if (-1 == ioctl(fd, VIDIOC_G_INPUT, &input.index)) {
    perror("VIDIOC_G_INPUT");
    exit(EXIT_FAILURE);
}

if (-1 == ioctl(fd, VIDIOC_ENUMINPUT, &input)) {
    perror("VIDIOC_ENUM_INPUT");
    exit(EXIT_FAILURE);
}

if (0 == (input.std & V4L2_STD_PAL_BG)) {
    fprintf(stderr, "Oops. B/G PAL is not supported.\\n");
    exit(EXIT_FAILURE);
}

/* Note this is also supposed to work when only B
   or G/PAL is supported. */

std_id = V4L2_STD_PAL_BG;
```

```
if (-1 == ioctl(fd, VIDIOC_S_STD, &std_id)) {
    perror("VIDIOC_S_STD");
    exit(EXIT_FAILURE);
}
```

Digital Video (DV) Timings

The video standards discussed so far have been dealing with Analog TV and the corresponding video timings. Today there are many more different hardware interfaces such as High Definition TV interfaces (HDMI), VGA, DVI connectors etc., that carry video signals and there is a need to extend the API to select the video timings for these interfaces. Since it is not possible to extend the *v4l2_std_id* due to the limited bits available, a new set of ioctls was added to set/get video timings at the input and output.

These ioctls deal with the detailed digital video timings that define each video format. This includes parameters such as the active video width and height, signal polarities, frontporches, backporches, sync widths etc. The `linux/v4l2-dv-timings.h` header can be used to get the timings of the formats in the *CEA-861-E* and *VESA DMT* standards.

To enumerate and query the attributes of the DV timings supported by a device applications use the *ioctl VIDIOC_ENUM_DV_TIMINGS, VIDIOC_SUBDEV_ENUM_DV_TIMINGS* and *ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP* ioctls. To set DV timings for the device applications use the *VIDIOC_S_DV_TIMINGS* ioctl and to get current DV timings they use the *VIDIOC_G_DV_TIMINGS* ioctl. To detect the DV timings as seen by the video receiver applications use the *ioctl VIDIOC_QUERY_DV_TIMINGS* ioctl.

Applications can make use of the *Input capabilities* and *Output capabilities* flags to determine whether the digital video ioctls can be used with the given input or output.

User Controls

Devices typically have a number of user-settable controls such as brightness, saturation and so on, which would be presented to the user on a graphical user interface. But, different devices will have different controls available, and furthermore, the range of possible values, and the default value will vary from device to device. The control ioctls provide the information and a mechanism to create a nice user interface for these controls that will work correctly with any device.

All controls are accessed using an ID value. V4L2 defines several IDs for specific purposes. Drivers can also implement their own custom controls using `V4L2_CID_PRIVATE_BASE` [1] and higher values. The predefined control IDs have the prefix `V4L2_CID_`, and are listed in *Control IDs* . The ID is used when querying the attributes of a control, and when getting or setting the current value.

Generally applications should present controls to the user without assumptions about their purpose. Each control comes with a name string the user is supposed to understand. When the purpose is non-intuitive the driver writer should provide a user manual, a user interface plug-in or a driver specific panel application. Predefined IDs were introduced to change a few controls programmatically, for example to mute a device during a channel switch.

Drivers may enumerate different controls after switching the current video input or output, tuner or modulator, or audio input or output. Different in the sense of other bounds, another default and current value, step size or other menu items. A control with a certain *custom* ID can also change name and type.

If a control is not applicable to the current configuration of the device (for example, it doesn't apply to the current video input) drivers set the `V4L2_CTRL_FLAG_INACTIVE` flag.

[1] The use of `V4L2_CID_PRIVATE_BASE` is problematic because different drivers may use the same `V4L2_CID_PRIVATE_BASE` ID for different controls. This makes it hard to programatically set such controls since the meaning of the control with that ID is driver dependent. In order to resolve this drivers use unique IDs and the `V4L2_CID_PRIVATE_BASE` IDs are mapped to those unique IDs by the kernel. Consider these `V4L2_CID_PRIVATE_BASE` IDs as aliases to the real IDs.

Many applications today still use the `V4L2_CID_PRIVATE_BASE` IDs instead of using *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* with the `V4L2_CTRL_FLAG_NEXT_CTRL` flag to enumerate all IDs, so support for `V4L2_CID_PRIVATE_BASE` is still around.

Control values are stored globally, they do not change when switching except to stay within the reported bounds. They also do not change e. g. when the device is opened or closed, when the tuner radio frequency is changed or generally never without application request.

V4L2 specifies an event mechanism to notify applications when controls change value (see *ioctl VIDIOC_SUBSCRIBE_EVENT, VIDIOC_UNSUBSCRIBE_EVENT* , event V4L2_EVENT_CTRL), panel applications might want to make use of that in order to always reflect the correct control value.

All controls use machine endianness.

Control IDs

V4L2_CID_BASE First predefined ID, equal to V4L2_CID_BRIGHTNESS.

V4L2_CID_USER_BASE Synonym of V4L2_CID_BASE.

V4L2_CID_BRIGHTNESS (integer) Picture brightness, or more precisely, the black level.

V4L2_CID_CONTRAST (integer) Picture contrast or luma gain.

V4L2_CID_SATURATION (integer) Picture color saturation or chroma gain.

V4L2_CID_HUE (integer) Hue or color balance.

V4L2_CID_AUDIO_VOLUME (integer) Overall audio volume. Note some drivers also provide an OSS or ALSA mixer interface.

V4L2_CID_AUDIO_BALANCE (integer) Audio stereo balance. Minimum corresponds to all the way left, maximum to right.

V4L2_CID_AUDIO_BASS (integer) Audio bass adjustment.

V4L2_CID_AUDIO_TREBLE (integer) Audio treble adjustment.

V4L2_CID_AUDIO_MUTE (boolean) Mute audio, i. e. set the volume to zero, however without affecting V4L2_CID_AUDIO_VOLUME. Like ALSA drivers, V4L2 drivers must mute at load time to avoid excessive noise. Actually the entire device should be reset to a low power consumption state.

V4L2_CID_AUDIO_LOUDNESS (boolean) Loudness mode (bass boost).

V4L2_CID_BLACK_LEVEL (integer) Another name for brightness (not a synonym of V4L2_CID_BRIGHTNESS). This control is deprecated and should not be used in new drivers and applications.

V4L2_CID_AUTO_WHITE_BALANCE (boolean) Automatic white balance (cameras).

V4L2_CID_DO_WHITE_BALANCE (button) This is an action control. When set (the value is ignored), the device will do a white balance and then hold the current setting. Contrast this with the boolean V4L2_CID_AUTO_WHITE_BALANCE, which, when activated, keeps adjusting the white balance.

V4L2_CID_RED_BALANCE (integer) Red chroma balance.

V4L2_CID_BLUE_BALANCE (integer) Blue chroma balance.

V4L2_CID_GAMMA (integer) Gamma adjust.

V4L2_CID_WHITENESS (integer) Whiteness for grey-scale devices. This is a synonym for V4L2_CID_GAMMA. This control is deprecated and should not be used in new drivers and applications.

V4L2_CID_EXPOSURE (integer) Exposure (cameras). [Unit?]

V4L2_CID_AUTOGAIN (boolean) Automatic gain/exposure control.

V4L2_CID_GAIN (integer) Gain control.

V4L2_CID_HFLIP (boolean) Mirror the picture horizontally.

V4L2_CID_VFLIP (boolean) Mirror the picture vertically.

V4L2_CID_POWER_LINE_FREQUENCY (enum) Enables a power line frequency filter to avoid flicker. Possible values for enum v4l2_power_line_frequency are: V4L2_CID_POWER_LINE_FREQUENCY_DISABLED (0), V4L2_CID_POWER_LINE_FREQUENCY_50HZ (1), V4L2_CID_POWER_LINE_FREQUENCY_60HZ (2) and V4L2_CID_POWER_LINE_FREQUENCY_AUTO (3).

V4L2_CID_HUE_AUTO (boolean) Enables automatic hue control by the device. The effect of setting V4L2_CID_HUE while automatic hue control is enabled is undefined, drivers should ignore such request.

V4L2_CID_WHITE_BALANCE_TEMPERATURE (integer) This control specifies the white balance settings as a color temperature in Kelvin. A driver should have a minimum of 2800 (incandescent) to 6500 (daylight). For more information about color temperature see Wikipedia.

V4L2_CID_SHARPNESS (integer) Adjusts the sharpness filters in a camera. The minimum value disables the filters, higher values give a sharper picture.

V4L2_CID_BACKLIGHT_COMPENSATION (integer) Adjusts the backlight compensation in a camera. The minimum value disables backlight compensation.

V4L2_CID_CHROMA_AGC (boolean) Chroma automatic gain control.

V4L2_CID_CHROMA_GAIN (integer) Adjusts the Chroma gain control (for use when chroma AGC is disabled).

V4L2_CID_COLOR_KILLER (boolean) Enable the color killer (i. e. force a black & white image in case of a weak video signal).

V4L2_CID_COLORFX (enum) Selects a color effect. The following values are defined:

V4L2_COLORFX_NONE	Color effect is disabled.
V4L2_COLORFX_ANTIQUE	An aging (old photo) effect.
V4L2_COLORFX_ART_FREEZE	Frost color effect.
V4L2_COLORFX_AQUA	Water color, cool tone.
V4L2_COLORFX_BW	Black and white.
V4L2_COLORFX_EMBOSS	Emboss, the highlights and shadows replace light/dark boundaries and low contrast areas are set to a gray background.
V4L2_COLORFX_GRASS_GREEN	Grass green.
V4L2_COLORFX_NEGATIVE	Negative.
V4L2_COLORFX_SEPIA	Sepia tone.
V4L2_COLORFX_SKETCH	Sketch.
V4L2_COLORFX_SKIN_WHITEN	Skin whiten.
V4L2_COLORFX_SKY_BLUE	Sky blue.
V4L2_COLORFX_SOLARIZATION	Solarization, the image is partially reversed in tone, only color values above or below a certain threshold are inverted.
V4L2_COLORFX_SILHOUETTE	Silhouette (outline).
V4L2_COLORFX_VIVID	Vivid colors.
V4L2_COLORFX_SET_CBCR	The Cb and Cr chroma components are replaced by fixed coefficients determined by V4L2_CID_COLORFX_CBCR control.

V4L2_CID_COLORFX_CBCR (integer) Determines the Cb and Cr coefficients for V4L2_COLORFX_SET_CBCR color effect. Bits [7:0] of the supplied 32 bit value are interpreted as Cr component, bits [15:8] as Cb component and bits [31:16] must be zero.

V4L2_CID_AUTOBRIGHTNESS (boolean) Enable Automatic Brightness.

V4L2_CID_ROTATE (integer) Rotates the image by specified angle. Common angles are 90, 270 and 180. Rotating the image to 90 and 270 will reverse the height and width of the display window. It is necessary to set the new height and width of the picture using the *VIDIOC_S_FMT* ioctl according to the rotation angle selected.

V4L2_CID_BG_COLOR (integer) Sets the background color on the current output device. Background color needs to be specified in the RGB24 format. The supplied 32 bit value is interpreted as bits 0-7 Red color information, bits 8-15 Green color information, bits 16-23 Blue color information and bits 24-31 must be zero.

V4L2_CID_ILLUMINATORS_1 V4L2_CID_ILLUMINATORS_2 (boolean) Switch on or off the illuminator 1 or 2 of the device (usually a microscope).

V4L2_CID_MIN_BUFFERS_FOR_CAPTURE (integer) This is a read-only control that can be read by the application and used as a hint to determine the number of CAPTURE buffers to pass to REQBUFS. The value is the minimum number of CAPTURE buffers that is necessary for hardware to work.

V4L2_CID_MIN_BUFFERS_FOR_OUTPUT (integer) This is a read-only control that can be read by the application and used as a hint to determine the number of OUTPUT buffers to pass to REQBUFS. The value is the minimum number of OUTPUT buffers that is necessary for hardware to work.

V4L2_CID_ALPHA_COMPONENT (integer) Sets the alpha color component. When a capture device (or capture queue of a mem-to-mem device) produces a frame format that includes an alpha component (e.g. *packed RGB image formats*) and the alpha value is not defined by the device or the mem-to-mem input data this control lets you select the alpha component value of all pixels. When an output device (or output queue of a mem-to-mem device) consumes a frame format that doesn't include an alpha component and the device supports alpha channel processing this control lets you set the alpha component value of all pixels for further processing in the device.

V4L2_CID_LASTP1 End of the predefined control IDs (currently V4L2_CID_ALPHA_COMPONENT + 1).

V4L2_CID_PRIVATE_BASE ID of the first custom (driver specific) control. Applications depending on particular custom controls should check the driver name and version, see *Querying Capabilities* .

Applications can enumerate the available controls with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* and *VIDIOC_QUERYMENU* ioctls, get and set a control value with the *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls. Drivers must implement VIDIOC_QUERYCTRL, VIDIOC_G_CTRL and VIDIOC_S_CTRL when the device has one or more controls, VIDIOC_QUERYMENU when it has one or more menu type controls.

Example: Enumerating all controls

```
struct v4l2_queryctrl queryctrl;
struct v4l2_querymenu querymenu;

static void enumerate_menu(__u32 id)
{
    printf("  Menu items:\\n");

    memset(&querymenu, 0, sizeof(querymenu));
    querymenu.id = id;

    for (querymenu.index = queryctrl.minimum;
         querymenu.index <= queryctrl.maximum;
         querymenu.index++) {
        if (0 == ioctl(fd, VIDIOC_QUERYMENU, &querymenu)) {
            printf("  %s\\n", querymenu.name);
        }
    }
}

memset(&queryctrl, 0, sizeof(queryctrl));

queryctrl.id = V4L2_CTRL_FLAG_NEXT_CTRL;
while (0 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
    if (!(queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)) {
        printf("Control %s\\n", queryctrl.name);

        if (queryctrl.type == V4L2_CTRL_TYPE_MENU)
            enumerate_menu(queryctrl.id);
    }
```

```
    queryctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL;
}
if (errno != EINVAL) {
    perror("VIDIOC_QUERYCTRL");
    exit(EXIT_FAILURE);
}
```

Example: Enumerating all controls including compound controls

```
struct v4l2_query_ext_ctrl query_ext_ctrl;

memset(&query_ext_ctrl, 0, sizeof(query_ext_ctrl));

query_ext_ctrl.id = V4L2_CTRL_FLAG_NEXT_CTRL | V4L2_CTRL_FLAG_NEXT_COMPOUND;
while (0 == ioctl(fd, VIDIOC_QUERY_EXT_CTRL, &query_ext_ctrl)) {
    if (!(query_ext_ctrl.flags & V4L2_CTRL_FLAG_DISABLED)) {
        printf("Control %s\\n", query_ext_ctrl.name);

        if (query_ext_ctrl.type == V4L2_CTRL_TYPE_MENU)
            enumerate_menu(query_ext_ctrl.id);
    }

    query_ext_ctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL | V4L2_CTRL_FLAG_NEXT_COMPOUND;
}
if (errno != EINVAL) {
    perror("VIDIOC_QUERY_EXT_CTRL");
    exit(EXIT_FAILURE);
}
```

Example: Enumerating all user controls (old style)

```
memset(&queryctrl, 0, sizeof(queryctrl));

for (queryctrl.id = V4L2_CID_BASE;
     queryctrl.id < V4L2_CID_LASTP1;
     queryctrl.id++) {
    if (0 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
        if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)
            continue;

        printf("Control %s\\n", queryctrl.name);

        if (queryctrl.type == V4L2_CTRL_TYPE_MENU)
            enumerate_menu(queryctrl.id);
    } else {
        if (errno == EINVAL)
            continue;

        perror("VIDIOC_QUERYCTRL");
        exit(EXIT_FAILURE);
    }
}

for (queryctrl.id = V4L2_CID_PRIVATE_BASE;;
     queryctrl.id++) {
    if (0 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
        if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)
            continue;
```

```
        printf("Control %s\\n", queryctrl.name);

        if (queryctrl.type == V4L2_CTRL_TYPE_MENU)
            enumerate_menu(queryctrl.id);
    } else {
        if (errno == EINVAL)
            break;

        perror("VIDIOC_QUERYCTRL");
        exit(EXIT_FAILURE);
    }
}
```

Example: Changing controls

```
struct v4l2_queryctrl queryctrl;
struct v4l2_control control;

memset(&queryctrl, 0, sizeof(queryctrl));
queryctrl.id = V4L2_CID_BRIGHTNESS;

if (-1 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
    if (errno != EINVAL) {
        perror("VIDIOC_QUERYCTRL");
        exit(EXIT_FAILURE);
    } else {
        printf("V4L2_CID_BRIGHTNESS is not supportedn");
    }
} else if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED) {
    printf("V4L2_CID_BRIGHTNESS is not supportedn");
} else {
    memset(&control, 0, sizeof (control));
    control.id = V4L2_CID_BRIGHTNESS;
    control.value = queryctrl.default_value;

    if (-1 == ioctl(fd, VIDIOC_S_CTRL, &control)) {
        perror("VIDIOC_S_CTRL");
        exit(EXIT_FAILURE);
    }
}

memset(&control, 0, sizeof(control));
control.id = V4L2_CID_CONTRAST;

if (0 == ioctl(fd, VIDIOC_G_CTRL, &control)) {
    control.value += 1;

    /* The driver may clamp the value or return ERANGE, ignored here */

    if (-1 == ioctl(fd, VIDIOC_S_CTRL, &control)
        && errno != ERANGE) {
        perror("VIDIOC_S_CTRL");
        exit(EXIT_FAILURE);
    }
/* Ignore if V4L2_CID_CONTRAST is unsupported */
} else if (errno != EINVAL) {
    perror("VIDIOC_G_CTRL");
    exit(EXIT_FAILURE);
}
```

```
control.id = V4L2_CID_AUDIO_MUTE;
control.value = 1; /* silence */

/* Errors ignored */
ioctl(fd, VIDIOC_S_CTRL, &control);
```

Extended Controls

Introduction

The control mechanism as originally designed was meant to be used for user settings (brightness, saturation, etc). However, it turned out to be a very useful model for implementing more complicated driver APIs where each driver implements only a subset of a larger API.

The MPEG encoding API was the driving force behind designing and implementing this extended control mechanism: the MPEG standard is quite large and the currently supported hardware MPEG encoders each only implement a subset of this standard. Further more, many parameters relating to how the video is encoded into an MPEG stream are specific to the MPEG encoding chip since the MPEG standard only defines the format of the resulting MPEG stream, not how the video is actually encoded into that format.

Unfortunately, the original control API lacked some features needed for these new uses and so it was extended into the (not terribly originally named) extended control API.

Even though the MPEG encoding API was the first effort to use the Extended Control API, nowadays there are also other classes of Extended Controls, such as Camera Controls and FM Transmitter Controls. The Extended Controls API as well as all Extended Controls classes are described in the following text.

The Extended Control API

Three new ioctls are available: *VIDIOC_G_EXT_CTRLS* , *VIDIOC_S_EXT_CTRLS* and *VIDIOC_TRY_EXT_CTRLS* . These ioctls act on arrays of controls (as opposed to the *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls that act on a single control). This is needed since it is often required to atomically change several controls at once.

Each of the new ioctls expects a pointer to a struct *v4l2_ext_controls*. This structure contains a pointer to the control array, a count of the number of controls in that array and a control class. Control classes are used to group similar controls into a single class. For example, control class V4L2_CTRL_CLASS_USER contains all user controls (i. e. all controls that can also be set using the old *VIDIOC_S_CTRL* ioctl). Control class V4L2_CTRL_CLASS_MPEG contains all controls relating to MPEG encoding, etc.

All controls in the control array must belong to the specified control class. An error is returned if this is not the case.

It is also possible to use an empty control array (count == 0) to check whether the specified control class is supported.

The control array is a struct *v4l2_ext_control* array. The struct *v4l2_ext_control* is very similar to struct *v4l2_control*, except for the fact that it also allows for 64-bit values and pointers to be passed.

Since the struct *v4l2_ext_control* supports pointers it is now also possible to have controls with compound types such as N-dimensional arrays and/or structures. You need to specify the V4L2_CTRL_FLAG_NEXT_COMPOUND when enumerating controls to actually be able to see such compound controls. In other words, these controls with compound types should only be used programmatically.

Since such compound controls need to expose more information about themselves than is possible with *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* the *VIDIOC_QUERY_EXT_CTRL* ioctl was added. In particular, this ioctl gives the dimensions of the N-dimensional array if this control consists of more than one element.

Note:

1. *It is important to realize that due to the flexibility of controls it is necessary to check whether the control you want to set actually is supported in the driver and what the valid range of values is. So use the* ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU *(or* VIDIOC_QUERY_EXT_CTRL *) and* VIDIOC_QUERYMENU *ioctls to check this.*
2. *It is possible that some of the menu indices in a control of type V4L2_CTRL_TYPE_MENU may not be supported (VIDIOC_QUERYMENU will return an error). A good example is the list of supported MPEG audio bitrates. Some drivers only support one or two bitrates, others support a wider range.*

All controls use machine endianness.

Enumerating Extended Controls

The recommended way to enumerate over the extended controls is by using *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* in combination with the V4L2_CTRL_FLAG_NEXT_CTRL flag:

```
struct v4l2_queryctrl qctrl;

qctrl.id = V4L2_CTRL_FLAG_NEXT_CTRL;
while (0 == ioctl (fd, VIDIOC_QUERYCTRL, &qctrl)) {
    /* ... */
    qctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL;
}
```

The initial control ID is set to 0 ORed with the V4L2_CTRL_FLAG_NEXT_CTRL flag. The VIDIOC_QUERYCTRL ioctl will return the first control with a higher ID than the specified one. When no such controls are found an error is returned.

If you want to get all controls within a specific control class, then you can set the initial qctrl.id value to the control class and add an extra check to break out of the loop when a control of another control class is found:

```
qctrl.id = V4L2_CTRL_CLASS_MPEG | V4L2_CTRL_FLAG_NEXT_CTRL;
while (0 == ioctl(fd, VIDIOC_QUERYCTRL, &qctrl)) {
    if (V4L2_CTRL_ID2CLASS(qctrl.id) != V4L2_CTRL_CLASS_MPEG)
        break;
        /* ... */
    qctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL;
}
```

The 32-bit qctrl.id value is subdivided into three bit ranges: the top 4 bits are reserved for flags (e. g. V4L2_CTRL_FLAG_NEXT_CTRL) and are not actually part of the ID. The remaining 28 bits form the control ID, of which the most significant 12 bits define the control class and the least significant 16 bits identify the control within the control class. It is guaranteed that these last 16 bits are always non-zero for controls. The range of 0x1000 and up are reserved for driver-specific controls. The macro V4L2_CTRL_ID2CLASS(id) returns the control class ID based on a control ID.

If the driver does not support extended controls, then VIDIOC_QUERYCTRL will fail when used in combination with V4L2_CTRL_FLAG_NEXT_CTRL. In that case the old method of enumerating control should be used (see *Example: Enumerating all controls*). But if it is supported, then it is guaranteed to enumerate over all controls, including driver-private controls.

Creating Control Panels

It is possible to create control panels for a graphical user interface where the user can select the various controls. Basically you will have to iterate over all controls using the method described above. Each control class starts with a control of type V4L2_CTRL_TYPE_CTRL_CLASS. VIDIOC_QUERYCTRL will return the name of this control class which can be used as the title of a tab page within a control panel.

The flags field of struct *v4l2_queryctrl* also contains hints on the behavior of the control. See the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* documentation for more details.

Codec Control Reference

Below all controls within the Codec control class are described. First the generic controls, then controls specific for certain hardware.

Note:

These controls are applicable to all codecs and not just MPEG. The defines are prefixed with V4L2_CID_MPEG/V4L2_MPEG as the controls were originally made for MPEG codecs and later extended to cover all encoding formats.

Generic Codec Controls

Codec Control IDs

V4L2_CID_MPEG_CLASS (class) The Codec class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class. This description can be used as the caption of a Tab page in a GUI, for example.

V4L2_CID_MPEG_STREAM_TYPE (enum)

enum v4l2_mpeg_stream_type - The MPEG-1, -2 or -4 output stream type. One cannot assume anything here. Each hardware MPEG encoder tends to support different subsets of the available MPEG stream types. This control is specific to multiplexed MPEG streams. The currently defined stream types are:

V4L2_MPEG_STREAM_TYPE_MPEG2_PS	MPEG-2 program stream
V4L2_MPEG_STREAM_TYPE_MPEG2_TS	MPEG-2 transport stream
V4L2_MPEG_STREAM_TYPE_MPEG1_SS	MPEG-1 system stream
V4L2_MPEG_STREAM_TYPE_MPEG2_DVD	MPEG-2 DVD-compatible stream
V4L2_MPEG_STREAM_TYPE_MPEG1_VCD	MPEG-1 VCD-compatible stream
V4L2_MPEG_STREAM_TYPE_MPEG2_SVCD	MPEG-2 SVCD-compatible stream

V4L2_CID_MPEG_STREAM_PID_PMT (integer) Program Map Table Packet ID for the MPEG transport stream (default 16)

V4L2_CID_MPEG_STREAM_PID_AUDIO (integer) Audio Packet ID for the MPEG transport stream (default 256)

V4L2_CID_MPEG_STREAM_PID_VIDEO (integer) Video Packet ID for the MPEG transport stream (default 260)

V4L2_CID_MPEG_STREAM_PID_PCR (integer) Packet ID for the MPEG transport stream carrying PCR fields (default 259)

V4L2_CID_MPEG_STREAM_PES_ID_AUDIO (integer) Audio ID for MPEG PES

V4L2_CID_MPEG_STREAM_PES_ID_VIDEO (integer) Video ID for MPEG PES

V4L2_CID_MPEG_STREAM_VBI_FMT (enum)

enum v4l2_mpeg_stream_vbi_fmt - Some cards can embed VBI data (e. g. Closed Caption, Teletext) into the MPEG stream. This control selects whether VBI data should be embedded, and if so, what embedding method should be used. The list of possible VBI formats depends on the driver. The currently defined VBI format types are:

V4L2_MPEG_STREAM_VBI_FMT_NONE	No VBI in the MPEG stream
V4L2_MPEG_STREAM_VBI_FMT_IVTV	VBI in private packets, IVTV format (documented in the kernel sources in the file Documentation/video4linux/cx2341x/README.vbi)

V4L2_CID_MPEG_AUDIO_SAMPLING_FREQ (enum)

enum v4l2_mpeg_audio_sampling_freq - MPEG Audio sampling frequency. Possible values are:

V4L2_MPEG_AUDIO_SAMPLING_FREQ_44100	44.1 kHz
V4L2_MPEG_AUDIO_SAMPLING_FREQ_48000	48 kHz
V4L2_MPEG_AUDIO_SAMPLING_FREQ_32000	32 kHz

V4L2_CID_MPEG_AUDIO_ENCODING (enum)

enum v4l2_mpeg_audio_encoding - MPEG Audio encoding. This control is specific to multiplexed MPEG streams. Possible values are:

V4L2_MPEG_AUDIO_ENCODING_LAYER_1	MPEG-1/2 Layer I encoding
V4L2_MPEG_AUDIO_ENCODING_LAYER_2	MPEG-1/2 Layer II encoding
V4L2_MPEG_AUDIO_ENCODING_LAYER_3	MPEG-1/2 Layer III encoding
V4L2_MPEG_AUDIO_ENCODING_AAC	MPEG-2/4 AAC (Advanced Audio Coding)
V4L2_MPEG_AUDIO_ENCODING_AC3	AC-3 aka ATSC A/52 encoding

V4L2_CID_MPEG_AUDIO_L1_BITRATE (enum)

enum v4l2_mpeg_audio_l1_bitrate - MPEG-1/2 Layer I bitrate. Possible values are:

V4L2_MPEG_AUDIO_L1_BITRATE_32K	32 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_64K	64 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_96K	96 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_128K	128 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_160K	160 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_192K	192 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_224K	224 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_256K	256 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_288K	288 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_320K	320 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_352K	352 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_384K	384 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_416K	416 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_448K	448 kbit/s

V4L2_CID_MPEG_AUDIO_L2_BITRATE (enum)

enum v4l2_mpeg_audio_l2_bitrate - MPEG-1/2 Layer II bitrate. Possible values are:

V4L2_MPEG_AUDIO_L2_BITRATE_32K	32 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_48K	48 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_56K	56 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_64K	64 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_80K	80 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_96K	96 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_112K	112 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_128K	128 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_160K	160 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_192K	192 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_224K	224 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_256K	256 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_320K	320 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_384K	384 kbit/s

V4L2_CID_MPEG_AUDIO_L3_BITRATE (enum)

enum v4l2_mpeg_audio_l3_bitrate - MPEG-1/2 Layer III bitrate. Possible values are:

V4L2_MPEG_AUDIO_L3_BITRATE_32K	32 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_40K	40 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_48K	48 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_56K	56 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_64K	64 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_80K	80 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_96K	96 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_112K	112 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_128K	128 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_160K	160 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_192K	192 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_224K	224 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_256K	256 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_320K	320 kbit/s

V4L2_CID_MPEG_AUDIO_AAC_BITRATE (integer) AAC bitrate in bits per second.

V4L2_CID_MPEG_AUDIO_AC3_BITRATE (enum)

enum v4l2_mpeg_audio_ac3_bitrate - AC-3 bitrate. Possible values are:

V4L2_MPEG_AUDIO_AC3_BITRATE_32K	32 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_40K	40 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_48K	48 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_56K	56 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_64K	64 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_80K	80 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_96K	96 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_112K	112 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_128K	128 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_160K	160 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_192K	192 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_224K	224 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_256K	256 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_320K	320 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_384K	384 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_448K	448 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_512K	512 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_576K	576 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_640K	640 kbit/s

V4L2_CID_MPEG_AUDIO_MODE (enum)

enum v4l2_mpeg_audio_mode - MPEG Audio mode. Possible values are:

V4L2_MPEG_AUDIO_MODE_STEREO	Stereo
V4L2_MPEG_AUDIO_MODE_JOINT_STEREO	Joint Stereo
V4L2_MPEG_AUDIO_MODE_DUAL	Bilingual
V4L2_MPEG_AUDIO_MODE_MONO	Mono

V4L2_CID_MPEG_AUDIO_MODE_EXTENSION (enum)

enum v4l2_mpeg_audio_mode_extension - Joint Stereo audio mode extension. In Layer I and II they indicate which subbands are in intensity stereo. All other subbands are coded in stereo. Layer III is not (yet) supported. Possible values are:

V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_4	Subbands 4-31 in intensity stereo
V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_8	Subbands 8-31 in intensity stereo
V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_12	Subbands 12-31 in intensity stereo
V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_16	Subbands 16-31 in intensity stereo

V4L2_CID_MPEG_AUDIO_EMPHASIS (enum)

enum v4l2_mpeg_audio_emphasis - Audio Emphasis. Possible values are:

V4L2_MPEG_AUDIO_EMPHASIS_NONE	None
V4L2_MPEG_AUDIO_EMPHASIS_50_DIV_15_uS	50/15 microsecond emphasis
V4L2_MPEG_AUDIO_EMPHASIS_CCITT_J17	CCITT J.17

V4L2_CID_MPEG_AUDIO_CRC (enum)

enum v4l2_mpeg_audio_crc - CRC method. Possible values are:

V4L2_MPEG_AUDIO_CRC_NONE	None
V4L2_MPEG_AUDIO_CRC_CRC16	16 bit parity check

V4L2_CID_MPEG_AUDIO_MUTE (boolean) Mutes the audio when capturing. This is not done by muting audio hardware, which can still produce a slight hiss, but in the encoder itself, guaranteeing a fixed and reproducible audio bitstream. 0 = unmuted, 1 = muted.

V4L2_CID_MPEG_AUDIO_DEC_PLAYBACK (enum)

enum v4l2_mpeg_audio_dec_playback - Determines how monolingual audio should be played back. Possible values are:

V4L2_MPEG_AUDIO_DEC_PLAYBACK_AUTO	Automatically determines the best playback mode.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_STEREO	Stereo playback.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_LEFT	Left channel playback.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_RIGHT	Right channel playback.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_MONO	Mono playback.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_SWAPPED_STEREO	Stereo playback with swapped left and right channels.

V4L2_CID_MPEG_AUDIO_DEC_MULTILINGUAL_PLAYBACK (enum)

enum v4l2_mpeg_audio_dec_playback - Determines how multilingual audio should be played back.

V4L2_CID_MPEG_VIDEO_ENCODING (enum)

enum v4l2_mpeg_video_encoding - MPEG Video encoding method. This control is specific to multiplexed MPEG streams. Possible values are:

V4L2_MPEG_VIDEO_ENCODING_MPEG_1	MPEG-1 Video encoding
V4L2_MPEG_VIDEO_ENCODING_MPEG_2	MPEG-2 Video encoding
V4L2_MPEG_VIDEO_ENCODING_MPEG_4_AVC	MPEG-4 AVC (H.264) Video encoding

V4L2_CID_MPEG_VIDEO_ASPECT (enum)

enum v4l2_mpeg_video_aspect - Video aspect. Possible values are:

V4L2_MPEG_VIDEO_ASPECT_1x1	
V4L2_MPEG_VIDEO_ASPECT_4x3	
V4L2_MPEG_VIDEO_ASPECT_16x9	
V4L2_MPEG_VIDEO_ASPECT_221x100	

V4L2_CID_MPEG_VIDEO_B_FRAMES (integer) Number of B-Frames (default 2)

V4L2_CID_MPEG_VIDEO_GOP_SIZE (integer) GOP size (default 12)

V4L2_CID_MPEG_VIDEO_GOP_CLOSURE (boolean) GOP closure (default 1)

V4L2_CID_MPEG_VIDEO_PULLDOWN (boolean) Enable 3:2 pulldown (default 0)

V4L2_CID_MPEG_VIDEO_BITRATE_MODE (enum)

enum v4l2_mpeg_video_bitrate_mode - Video bitrate mode. Possible values are:

V4L2_MPEG_VIDEO_BITRATE_MODE_VBR	Variable bitrate
V4L2_MPEG_VIDEO_BITRATE_MODE_CBR	Constant bitrate

V4L2_CID_MPEG_VIDEO_BITRATE (integer) Video bitrate in bits per second.

V4L2_CID_MPEG_VIDEO_BITRATE_PEAK (integer) Peak video bitrate in bits per second. Must be larger or equal to the average video bitrate. It is ignored if the video bitrate mode is set to constant bitrate.

V4L2_CID_MPEG_VIDEO_TEMPORAL_DECIMATION (integer) For every captured frame, skip this many subsequent frames (default 0).

V4L2_CID_MPEG_VIDEO_MUTE (boolean) "Mutes" the video to a fixed color when capturing. This is useful for testing, to produce a fixed video bitstream. 0 = unmuted, 1 = muted.

V4L2_CID_MPEG_VIDEO_MUTE_YUV (integer) Sets the "mute" color of the video. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:7	V chrominance information
Bit 8:15	U chrominance information
Bit 16:23	Y luminance information
Bit 24:31	Must be zero.

V4L2_CID_MPEG_VIDEO_DEC_PTS (integer64) This read-only control returns the 33-bit video Presentation Time Stamp as defined in ITU T-REC-H.222.0 and ISO/IEC 13818-1 of the currently displayed frame. This is the same PTS as is used in *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* .

V4L2_CID_MPEG_VIDEO_DEC_FRAME (integer64) This read-only control returns the frame counter of the frame that is currently displayed (decoded). This value is reset to 0 whenever the decoder is started.

V4L2_CID_MPEG_VIDEO_DECODER_SLICE_INTERFACE (boolean) If enabled the decoder expects to receive a single slice per buffer, otherwise the decoder expects a single frame in per buffer. Applicable to the decoder, all codecs.

V4L2_CID_MPEG_VIDEO_H264_VUI_SAR_ENABLE (boolean) Enable writing sample aspect ratio in the Video Usability Information. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_VUI_SAR_IDC (enum)

enum v4l2_mpeg_video_h264_vui_sar_idc - VUI sample aspect ratio indicator for H.264 encoding. The value is defined in the table E-1 in the standard. Applicable to the H264 encoder.

V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_UNSPECIFIED	Unspecified
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_1x1	1x1
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_12x11	12x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_10x11	10x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_16x11	16x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_40x33	40x33
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_24x11	24x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_20x11	20x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_32x11	32x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_80x33	80x33
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_18x11	18x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_15x11	15x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_64x33	64x33
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_160x99	160x99
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_4x3	4x3
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_3x2	3x2
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_2x1	2x1
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_EXTENDED	Extended SAR

V4L2_CID_MPEG_VIDEO_H264_VUI_EXT_SAR_WIDTH (integer) Extended sample aspect ratio width for H.264 VUI encoding. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_VUI_EXT_SAR_HEIGHT (integer) Extended sample aspect ratio height for H.264 VUI encoding. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_LEVEL (enum)

enum v4l2_mpeg_video_h264_level - The level information for the H264 video elementary stream. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_LEVEL_1_0	Level 1.0
V4L2_MPEG_VIDEO_H264_LEVEL_1B	Level 1B
V4L2_MPEG_VIDEO_H264_LEVEL_1_1	Level 1.1
V4L2_MPEG_VIDEO_H264_LEVEL_1_2	Level 1.2
V4L2_MPEG_VIDEO_H264_LEVEL_1_3	Level 1.3
V4L2_MPEG_VIDEO_H264_LEVEL_2_0	Level 2.0
V4L2_MPEG_VIDEO_H264_LEVEL_2_1	Level 2.1
V4L2_MPEG_VIDEO_H264_LEVEL_2_2	Level 2.2
V4L2_MPEG_VIDEO_H264_LEVEL_3_0	Level 3.0
V4L2_MPEG_VIDEO_H264_LEVEL_3_1	Level 3.1
V4L2_MPEG_VIDEO_H264_LEVEL_3_2	Level 3.2
V4L2_MPEG_VIDEO_H264_LEVEL_4_0	Level 4.0
V4L2_MPEG_VIDEO_H264_LEVEL_4_1	Level 4.1
V4L2_MPEG_VIDEO_H264_LEVEL_4_2	Level 4.2
V4L2_MPEG_VIDEO_H264_LEVEL_5_0	Level 5.0
V4L2_MPEG_VIDEO_H264_LEVEL_5_1	Level 5.1

V4L2_CID_MPEG_VIDEO_MPEG4_LEVEL (enum)

enum v4l2_mpeg_video_mpeg4_level - The level information for the MPEG4 elementary stream. Applicable to the MPEG4 encoder. Possible values are:

V4L2_MPEG_VIDEO_LEVEL_0	Level 0
V4L2_MPEG_VIDEO_LEVEL_0B	Level 0b
V4L2_MPEG_VIDEO_LEVEL_1	Level 1
V4L2_MPEG_VIDEO_LEVEL_2	Level 2
V4L2_MPEG_VIDEO_LEVEL_3	Level 3
V4L2_MPEG_VIDEO_LEVEL_3B	Level 3b
V4L2_MPEG_VIDEO_LEVEL_4	Level 4
V4L2_MPEG_VIDEO_LEVEL_5	Level 5

V4L2_CID_MPEG_VIDEO_H264_PROFILE (enum)

enum v4l2_mpeg_video_h264_profile - The profile information for H264. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_PROFILE_BASELINE	Baseline profile
V4L2_MPEG_VIDEO_H264_PROFILE_CONSTRAINED_BASELINE	Constrained Baseline profile
V4L2_MPEG_VIDEO_H264_PROFILE_MAIN	Main profile
V4L2_MPEG_VIDEO_H264_PROFILE_EXTENDED	Extended profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH	High profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_10	High 10 profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_422	High 422 profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_444_PREDICTIVE	High 444 Predictive profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_10_INTRA	High 10 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_422_INTRA	High 422 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_444_INTRA	High 444 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_CAVLC_444_INTRA	CAVLC 444 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_SCALABLE_BASELINE	Scalable Baseline profile
V4L2_MPEG_VIDEO_H264_PROFILE_SCALABLE_HIGH	Scalable High profile
V4L2_MPEG_VIDEO_H264_PROFILE_SCALABLE_HIGH_INTRA	Scalable High Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_STEREO_HIGH	Stereo High profile
V4L2_MPEG_VIDEO_H264_PROFILE_MULTIVIEW_HIGH	Multiview High profile

V4L2_CID_MPEG_VIDEO_MPEG4_PROFILE (enum)

enum v4l2_mpeg_video_mpeg4_profile - The profile information for MPEG4. Applicable to the MPEG4 encoder. Possible values are:

V4L2_MPEG_VIDEO_PROFILE_SIMPLE	Simple profile
V4L2_MPEG_VIDEO_PROFILE_ADVANCED_SIMPLE	Advanced Simple profile
V4L2_MPEG_VIDEO_PROFILE_CORE	Core profile
V4L2_MPEG_VIDEO_PROFILE_SIMPLE_SCALABLE	Simple Scalable profile
V4L2_MPEG_VIDEO_PROFILE_ADVANCED_CODING_EFFICIENCY	

V4L2_CID_MPEG_VIDEO_MAX_REF_PIC (integer) The maximum number of reference pictures used for encoding. Applicable to the encoder.

V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE (enum)

enum v4l2_mpeg_video_multi_slice_mode - Determines how the encoder should handle division of frame into slices. Applicable to the encoder. Possible values are:

V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_SINGLE	Single slice per frame.
V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_MB	Multiple slices with set maximum number of macroblocks per slice.
V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_BYTES	Multiple slice with set maximum size in bytes per slice.

V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_MB (integer) The maximum number of macroblocks in a slice. Used when V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE is set to V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_MB. Applicable to the encoder.

V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_BYTES (integer) The maximum size of a slice in bytes. Used when V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE is set to V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_BYTES. Applicable to the encoder.

V4L2_CID_MPEG_VIDEO_H264_LOOP_FILTER_MODE (enum)

enum v4l2_mpeg_video_h264_loop_filter_mode - Loop filter mode for H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_LOOP_FILTER_MODE_ENABLED	Loop filter is enabled.
V4L2_MPEG_VIDEO_H264_LOOP_FILTER_MODE_DISABLED	Loop filter is disabled.
V4L2_MPEG_VIDEO_H264_LOOP_FILTER_MODE_DISABLED_AT_SLICE_BOUNDARY	Loop filter is disabled at the slice boundary.

V4L2_CID_MPEG_VIDEO_H264_LOOP_FILTER_ALPHA (integer) Loop filter alpha coefficient, defined in the H264 standard. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_LOOP_FILTER_BETA (integer) Loop filter beta coefficient, defined in the H264 standard. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_ENTROPY_MODE (enum)

enum v4l2_mpeg_video_h264_entropy_mode - Entropy coding mode for H264 - CABAC/CAVALC. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_ENTROPY_MODE_CAVLC	Use CAVLC entropy coding.
V4L2_MPEG_VIDEO_H264_ENTROPY_MODE_CABAC	Use CABAC entropy coding.

V4L2_CID_MPEG_VIDEO_H264_8X8_TRANSFORM (boolean) Enable 8X8 transform for H264. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_CYCLIC_INTRA_REFRESH_MB (integer) Cyclic intra macroblock refresh. This is the number of continuous macroblocks refreshed every frame. Each frame a successive set of macroblocks is refreshed until the cycle completes and starts from the top of the frame. Applicable to H264, H263 and MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_FRAME_RC_ENABLE (boolean) Frame level rate control enable. If this control is disabled then the quantization parameter for each frame type is constant and set with appropriate controls (e.g. V4L2_CID_MPEG_VIDEO_H263_I_FRAME_QP). If frame rate control is enabled then quantization parameter is adjusted to meet the chosen bitrate. Minimum and maximum value for the quantization parameter can be set with appropriate controls (e.g. V4L2_CID_MPEG_VIDEO_H263_MIN_QP). Applicable to encoders.

V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE (boolean) Macroblock level rate control enable. Applicable to the MPEG4 and H264 encoders.

V4L2_CID_MPEG_VIDEO_MPEG4_QPEL (boolean) Quarter pixel motion estimation for MPEG4. Applicable to the MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_H263_I_FRAME_QP (integer) Quantization parameter for an I frame for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_MIN_QP (integer) Minimum quantization parameter for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_MAX_QP (integer) Maximum quantization parameter for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_P_FRAME_QP (integer) Quantization parameter for an P frame for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_B_FRAME_QP (integer) Quantization parameter for an B frame for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H264_I_FRAME_QP (integer) Quantization parameter for an I frame for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_MIN_QP (integer) Minimum quantization parameter for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_MAX_QP (integer) Maximum quantization parameter for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_P_FRAME_QP (integer) Quantization parameter for an P frame for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_B_FRAME_QP (integer) Quantization parameter for an B frame for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_MPEG4_I_FRAME_QP (integer) Quantization parameter for an I frame for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_MIN_QP (integer) Minimum quantization parameter for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_MAX_QP (integer) Maximum quantization parameter for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_P_FRAME_QP (integer) Quantization parameter for an P frame for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_B_FRAME_QP (integer) Quantization parameter for an B frame for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_VBV_SIZE (integer) The Video Buffer Verifier size in kilobytes, it is used as a limitation of frame skip. The VBV is defined in the standard as a mean to verify that the produced stream will be successfully decoded. The standard describes it as “Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.”. Applicable to the MPEG1, MPEG2, MPEG4 encoders.

V4L2_CID_MPEG_VIDEO_VBV_DELAY (integer) Sets the initial delay in milliseconds for VBV buffer control.

V4L2_CID_MPEG_VIDEO_MV_H_SEARCH_RANGE (integer) Horizontal search range defines maximum horizontal search area in pixels to search and match for the present Macroblock (MB) in the reference picture. This V4L2 control macro is used to set horizontal search range for motion estimation module in video encoder.

V4L2_CID_MPEG_VIDEO_MV_V_SEARCH_RANGE (integer) Vertical search range defines maximum vertical search area in pixels to search and match for the present Macroblock (MB) in the reference picture. This V4L2 control macro is used to set vertical search range for motion estimation module in video encoder.

V4L2_CID_MPEG_VIDEO_FORCE_KEY_FRAME (button) Force a key frame for the next queued buffer. Applicable to encoders. This is a general, codec-agnostic keyframe control.

V4L2_CID_MPEG_VIDEO_H264_CPB_SIZE (integer) The Coded Picture Buffer size in kilobytes, it is used as a limitation of frame skip. The CPB is defined in the H264 standard as a mean to verify that the produced stream will be successfully decoded. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_I_PERIOD (integer) Period between I-frames in the open GOP for H264. In case of an open GOP this is the period between two I-frames. The period between IDR (Instantaneous Decoding Refresh) frames is taken from the GOP_SIZE control. An IDR frame, which stands for Instantaneous Decoding Refresh is an I-frame after which no prior frames are referenced. This means that a stream can be restarted from an IDR frame without the need to store or decode any previous frames. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_HEADER_MODE (enum)

enum v4l2_mpeg_video_header_mode - Determines whether the header is returned as the first buffer or is it returned together with the first frame. Applicable to encoders. Possible values are:

V4L2_MPEG_VIDEO_HEADER_MODE_SEPARATE	The stream header is returned separately in the first buffer.
V4L2_MPEG_VIDEO_HEADER_MODE_JOINED_WITH_1ST_FRAME	The stream header is returned together with the first encoded frame.

V4L2_CID_MPEG_VIDEO_REPEAT_SEQ_HEADER (boolean) Repeat the video sequence headers. Repeating these headers makes random access to the video stream easier. Applicable to the MPEG1, 2 and 4

encoder.

V4L2_CID_MPEG_VIDEO_DECODER_MPEG4_DEBLOCK_FILTER (boolean) Enabled the deblocking post processing filter for MPEG4 decoder. Applicable to the MPEG4 decoder.

V4L2_CID_MPEG_VIDEO_MPEG4_VOP_TIME_RES (integer) vop_time_increment_resolution value for MPEG4. Applicable to the MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_MPEG4_VOP_TIME_INC (integer) vop_time_increment value for MPEG4. Applicable to the MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_H264_SEI_FRAME_PACKING (boolean) Enable generation of frame packing supplemental enhancement information in the encoded bitstream. The frame packing SEI message contains the arrangement of L and R planes for 3D viewing. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_SEI_FP_CURRENT_FRAME_0 (boolean) Sets current frame as frame0 in frame packing SEI. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE (enum)

enum v4l2_mpeg_video_h264_sei_fp_arrangement_type - Frame packing arrangement type for H264 SEI. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_CHEKERBOARD	Pixels are alternatively from L and R.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_COLUMN	L and R are interlaced by column.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_ROW	L and R are interlaced by row.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_SIDE_BY_SIDE	L is on the left, R on the right.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_TOP_BOTTOM	L is on top, R on bottom.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_TEMPORAL	One view per frame.

V4L2_CID_MPEG_VIDEO_H264_FMO (boolean) Enables flexible macroblock ordering in the encoded bitstream. It is a technique used for restructuring the ordering of macroblocks in pictures. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_FMO_MAP_TYPE (enum)

enum v4l2_mpeg_video_h264_fmo_map_type - When using FMO, the map type divides the image in different scan patterns of macroblocks. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_INTERLEAVED_SLICES	Slices are interleaved one after other with macroblocks in run length order.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_SCATTERED_SLICES	Scatters the macroblocks based on a mathematical function known to both encoder and decoder.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_FOREGROUND_WITH_LEFT_OVER	Macroblocks arranged in rectangular areas or regions of interest.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_BOX_OUT	Slice groups grow in a cyclic way from centre to outwards.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_RASTER_SCAN	Slice groups grow in raster scan pattern from left to right.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_WIPE_SCAN	Slice groups grow in wipe scan pattern from top to bottom.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_EXPLICIT	User defined map type.

V4L2_CID_MPEG_VIDEO_H264_FMO_SLICE_GROUP (integer) Number of slice groups in FMO. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_FMO_CHANGE_DIRECTION (enum)

enum v4l2_mpeg_video_h264_fmo_change_dir - Specifies a direction of the slice group change for raster and wipe maps. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_FMO_CHANGE_DIR_RIGHT	Raster scan or wipe right.
V4L2_MPEG_VIDEO_H264_FMO_CHANGE_DIR_LEFT	Reverse raster scan or wipe left.

V4L2_CID_MPEG_VIDEO_H264_FMO_CHANGE_RATE (integer) Specifies the size of the first slice group for raster and wipe map. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_FMO_RUN_LENGTH (integer) Specifies the number of consecutive macroblocks for the interleaved map. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_ASO (boolean) Enables arbitrary slice ordering in encoded bitstream. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_ASO_SLICE_ORDER (integer) Specifies the slice order in ASO. Applicable to the H264 encoder. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:15	Slice ID
Bit 16:32	Slice position or order

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING (boolean) Enables H264 hierarchical coding. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_TYPE (enum)

enum v4l2_mpeg_video_h264_hierarchical_coding_type - Specifies the hierarchical coding type. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_HIERARCHICAL_CODING_B	Hierarchical B coding.
V4L2_MPEG_VIDEO_H264_HIERARCHICAL_CODING_P	Hierarchical P coding.

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_LAYER (integer) Specifies the number of hierarchical coding layers. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_LAYER_QP (integer) Specifies a user defined QP for each layer. Applicable to the H264 encoder. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:15	QP value
Bit 16:32	Layer number

MFC 5.1 MPEG Controls

The following MPEG class controls deal with MPEG decoding and encoding settings that are specific to the Multi Format Codec 5.1 device present in the S5P family of SoCs by Samsung.

MFC 5.1 Control IDs

V4L2_CID_MPEG_MFC51_VIDEO_DECODER_H264_DISPLAY_DELAY_ENABLE (boolean) If the display delay is enabled then the decoder is forced to return a CAPTURE buffer (decoded frame) after processing a certain number of OUTPUT buffers. The delay can be set through V4L2_CID_MPEG_MFC51_VIDEO_DECODER_H264_DISPLAY_DELAY. This feature can be used for example for generating thumbnails of videos. Applicable to the H264 decoder.

V4L2_CID_MPEG_MFC51_VIDEO_DECODER_H264_DISPLAY_DELAY (integer) Display delay value for H264 decoder. The decoder is forced to return a decoded frame after the set 'display delay' number of frames. If this number is low it may result in frames returned out of dispaly order, in addition the hardware may still be using the returned buffer as a reference picture for subsequent frames.

V4L2_CID_MPEG_MFC51_VIDEO_H264_NUM_REF_PIC_FOR_P (integer) The number of reference pictures used for encoding a P picture. Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_PADDING (boolean) Padding enable in the encoder - use a color instead of repeating border pixels. Applicable to encoders.

V4L2_CID_MPEG_MFC51_VIDEO_PADDING_YUV (integer) Padding color in the encoder. Applicable to encoders. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:7	V chrominance information
Bit 8:15	U chrominance information
Bit 16:23	Y luminance information
Bit 24:31	Must be zero.

V4L2_CID_MPEG_MFC51_VIDEO_RC_REACTION_COEFF (integer) Reaction coefficient for MFC rate control. Applicable to encoders.

Note:

1. *Valid only when the frame level RC is enabled.*
2. *For tight CBR, this field must be small (ex. 2 ~ 10). For VBR, this field must be large (ex. 100 ~ 1000).*
3. *It is not recommended to use the greater number than FRAME_RATE * (10^9 / BIT_RATE).*

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_DARK (boolean) Adaptive rate control for dark region. Valid only when H.264 and macroblock level RC is enabled (V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_SMOOTH (boolean) Adaptive rate control for smooth region. Valid only when H.264 and macroblock level RC is enabled (V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_STATIC (boolean) Adaptive rate control for static region. Valid only when H.264 and macroblock level RC is enabled (V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_ACTIVITY (boolean) Adaptive rate control for activity region. Valid only when H.264 and macroblock level RC is enabled (V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_FRAME_SKIP_MODE (enum)

enum v4l2_mpeg_mfc51_video_frame_skip_mode - Indicates in what conditions the encoder should skip frames. If encoding a frame would cause the encoded stream to be larger then a chosen data limit then the frame will be skipped. Possible values are:

V4L2_MPEG_MFC51_FRAME_SKIP_MODE_DISABLED	Frame skip mode is disabled.
V4L2_MPEG_MFC51_FRAME_SKIP_MODE_LEVEL_LIMIT	Frame skip mode enabled and buffer limit is set by the chosen level and is defined by the standard.
V4L2_MPEG_MFC51_FRAME_SKIP_MODE_BUF_LIMIT	Frame skip mode enabled and buffer limit is set by the VBV (MPEG1/2/4) or CPB (H264) buffer size control.

V4L2_CID_MPEG_MFC51_VIDEO_RC_FIXED_TARGET_BIT (integer) Enable rate-control with fixed target bit. If this setting is enabled, then the rate control logic of the encoder will calculate the average bitrate for a GOP and keep it below or equal the set bitrate target. Otherwise the rate control logic calculates the overall average bitrate for the stream and keeps it below or equal to the set bitrate. In the first case the average bitrate for the whole stream will be smaller then the set bitrate. This is caused because the average is calculated for smaller number of frames, on the other hand enabling this setting will ensure that the stream will meet tight bandwidth constraints. Applicable to encoders.

V4L2_CID_MPEG_MFC51_VIDEO_FORCE_FRAME_TYPE (enum)

enum v4l2_mpeg_mfc51_video_force_frame_type - Force a frame type for the next queued buffer. Applicable to encoders. Possible values are:

V4L2_MPEG_MFC51_FORCE_FRAME_TYPE_DISABLED	Forcing a specific frame type disabled.
V4L2_MPEG_MFC51_FORCE_FRAME_TYPE_I_FRAME	Force an I-frame.
V4L2_MPEG_MFC51_FORCE_FRAME_TYPE_NOT_CODED	Force a non-coded frame.

CX2341x MPEG Controls

The following MPEG class controls deal with MPEG encoding settings that are specific to the Conexant CX23415 and CX23416 MPEG encoding chips.

CX2341x Control IDs

V4L2_CID_MPEG_CX2341X_VIDEO_SPATIAL_FILTER_MODE (enum)

enum v4l2_mpeg_cx2341x_video_spatial_filter_mode - Sets the Spatial Filter mode (default MANUAL). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_SPATIAL_FILTER_MODE_MANUAL	Choose the filter manually
V4L2_MPEG_CX2341X_VIDEO_SPATIAL_FILTER_MODE_AUTO	Choose the filter automatically

V4L2_CID_MPEG_CX2341X_VIDEO_SPATIAL_FILTER (integer (0-15)) The setting for the Spatial Filter. 0 = off, 15 = maximum. (Default is 0.)

V4L2_CID_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE (enum)

enum v4l2_mpeg_cx2341x_video_luma_spatial_filter_type - Select the algorithm to use for the Luma Spatial Filter (default 1D_HOR). Possible values:

V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_OFF	No filter
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_1D_HOR	One-dimensional horizontal
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_1D_VERT	One-dimensional vertical
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_2D_HV_SEPARABLE	Two-dimensional separable
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_2D_SYM_NON_SEPARABLE	Two-dimensional symmetrical non-separable

V4L2_CID_MPEG_CX2341X_VIDEO_CHROMA_SPATIAL_FILTER_TYPE (enum)

enum v4l2_mpeg_cx2341x_video_chroma_spatial_filter_type - Select the algorithm for the Chroma Spatial Filter (default 1D_HOR). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_CHROMA_SPATIAL_FILTER_TYPE_OFF	No filter
V4L2_MPEG_CX2341X_VIDEO_CHROMA_SPATIAL_FILTER_TYPE_1D_HOR	One-dimensional horizontal

V4L2_CID_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER_MODE (enum)

enum v4l2_mpeg_cx2341x_video_temporal_filter_mode - Sets the Temporal Filter mode (default MANUAL). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER_MODE_MANUAL	Choose the filter manually
V4L2_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER_MODE_AUTO	Choose the filter automatically

V4L2_CID_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER (integer (0-31)) The setting for the Temporal Filter. 0 = off, 31 = maximum. (Default is 8 for full-scale capturing and 0 for scaled capturing.)

V4L2_CID_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE (enum)

enum v4l2_mpeg_cx2341x_video_median_filter_type - Median Filter Type (default 0FF). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_OFF	No filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_HOR	Horizontal filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_VERT	Vertical filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_HOR_VERT	Horizontal and vertical filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_DIAG	Diagonal filter

V4L2_CID_MPEG_CX2341X_VIDEO_LUMA_MEDIAN_FILTER_BOTTOM (integer (0-255)) Threshold above which the luminance median filter is enabled (default 0)

V4L2_CID_MPEG_CX2341X_VIDEO_LUMA_MEDIAN_FILTER_TOP (integer (0-255)) Threshold below which the luminance median filter is enabled (default 255)

V4L2_CID_MPEG_CX2341X_VIDEO_CHROMA_MEDIAN_FILTER_BOTTOM (integer (0-255)) Threshold above which the chroma median filter is enabled (default 0)

V4L2_CID_MPEG_CX2341X_VIDEO_CHROMA_MEDIAN_FILTER_TOP (integer (0-255)) Threshold below which the chroma median filter is enabled (default 255)

V4L2_CID_MPEG_CX2341X_STREAM_INSERT_NAV_PACKETS (boolean) The CX2341X MPEG encoder can insert one empty MPEG-2 PES packet into the stream between every four video frames. The packet size is 2048 bytes, including the packet_start_code_prefix and stream_id fields. The stream_id is 0xBF (private stream 2). The payload consists of 0x00 bytes, to be filled in by the application. 0 = do not insert, 1 = insert packets.

VPX Control Reference

The VPX controls include controls for encoding parameters of VPx video codec.

VPX Control IDs

V4L2_CID_MPEG_VIDEO_VPX_NUM_PARTITIONS (enum)

enum v4l2_vp8_num_partitions - The number of token partitions to use in VP8 encoder. Possible values are:

V4L2_CID_MPEG_VIDEO_VPX_1_PARTITION	1 coefficient partition
V4L2_CID_MPEG_VIDEO_VPX_2_PARTITIONS	2 coefficient partitions
V4L2_CID_MPEG_VIDEO_VPX_4_PARTITIONS	4 coefficient partitions
V4L2_CID_MPEG_VIDEO_VPX_8_PARTITIONS	8 coefficient partitions

V4L2_CID_MPEG_VIDEO_VPX_IMD_DISABLE_4X4 (boolean) Setting this prevents intra 4x4 mode in the intra mode decision.

V4L2_CID_MPEG_VIDEO_VPX_NUM_REF_FRAMES (enum)

enum v4l2_vp8_num_ref_frames - The number of reference pictures for encoding P frames. Possible values are:

V4L2_CID_MPEG_VIDEO_VPX_1_REF_FRAME	Last encoded frame will be searched
V4L2_CID_MPEG_VIDEO_VPX_2_REF_FRAME	Two frames will be searched among the last encoded frame, the golden frame and the alternate reference (altref) frame. The encoder implementation will decide which two are chosen.
V4L2_CID_MPEG_VIDEO_VPX_3_REF_FRAME	The last encoded frame, the golden frame and the altref frame will be searched.

V4L2_CID_MPEG_VIDEO_VPX_FILTER_LEVEL (integer) Indicates the loop filter level. The adjustment of the loop filter level is done via a delta value against a baseline loop filter value.

V4L2_CID_MPEG_VIDEO_VPX_FILTER_SHARPNESS (integer) This parameter affects the loop filter. Anything above zero weakens the deblocking effect on the loop filter.

V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_REF_PERIOD (integer) Sets the refresh period for the golden frame. The period is defined in number of frames. For a value of 'n', every nth frame starting from the first key frame will be taken as a golden frame. For eg. for encoding sequence of 0, 1, 2, 3, 4, 5, 6, 7 where the golden frame refresh period is set as 4, the frames 0, 4, 8 etc will be taken as the golden frames as frame 0 is always a key frame.

V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_SEL (enum)

enum v4l2_vp8_golden_frame_sel - Selects the golden frame for encoding. Possible values are:

V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_USE_PREV	Use the (n-2)th frame as a golden frame, current frame index being 'n'.
V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_USE_REF_PERIOD	Use the previous specific frame indicated by V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_REF_PERIOD as a golden frame.

V4L2_CID_MPEG_VIDEO_VPX_MIN_QP (integer) Minimum quantization parameter for VP8.

V4L2_CID_MPEG_VIDEO_VPX_MAX_QP (integer) Maximum quantization parameter for VP8.

V4L2_CID_MPEG_VIDEO_VPX_I_FRAME_QP (integer) Quantization parameter for an I frame for VP8.

V4L2_CID_MPEG_VIDEO_VPX_P_FRAME_QP (integer) Quantization parameter for a P frame for VP8.

V4L2_CID_MPEG_VIDEO_VPX_PROFILE (integer) Select the desired profile for VPx encoder. Acceptable values are 0, 1, 2 and 3 corresponding to encoder profiles 0, 1, 2 and 3.

Camera Control Reference

The Camera class includes controls for mechanical (or equivalent digital) features of a device such as controllable lenses or sensors.

Camera Control IDs

V4L2_CID_CAMERA_CLASS (class) The Camera class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_EXPOSURE_AUTO (enum)

enum v4l2_exposure_auto_type - Enables automatic adjustments of the exposure time and/or iris aperture. The effect of manual changes of the exposure time or iris aperture while these features are enabled is undefined, drivers should ignore such requests. Possible values are:

V4L2_EXPOSURE_AUTO	Automatic exposure time, automatic iris aperture.
V4L2_EXPOSURE_MANUAL	Manual exposure time, manual iris.
V4L2_EXPOSURE_SHUTTER_PRIORITY	Manual exposure time, auto iris.
V4L2_EXPOSURE_APERTURE_PRIORITY	Auto exposure time, manual iris.

V4L2_CID_EXPOSURE_ABSOLUTE (integer) Determines the exposure time of the camera sensor. The exposure time is limited by the frame interval. Drivers should interpret the values as 100 µs units, where the value 1 stands for 1/10000th of a second, 10000 for 1 second and 100000 for 10 seconds.

V4L2_CID_EXPOSURE_AUTO_PRIORITY (boolean) When V4L2_CID_EXPOSURE_AUTO is set to AUTO or APERTURE_PRIORITY, this control determines if the device may dynamically vary the frame rate. By default this feature is disabled (0) and the frame rate must remain constant.

V4L2_CID_EXPOSURE_BIAS (integer menu) Determines the automatic exposure compensation, it is effective only when V4L2_CID_EXPOSURE_AUTO control is set to AUTO, SHUTTER_PRIORITY or APERTURE_PRIORITY. It is expressed in terms of EV, drivers should interpret the values as 0.001 EV units, where the value 1000 stands for +1 EV.

Increasing the exposure compensation value is equivalent to decreasing the exposure value (EV) and will increase the amount of light at the image sensor. The camera performs the exposure compensation by adjusting absolute exposure time and/or aperture.

V4L2_CID_EXPOSURE_METERING (enum)

enum v4l2_exposure_metering - Determines how the camera measures the amount of light available for the frame exposure. Possible values are:

V4L2_EXPOSURE_METERING_AVERAGE	Use the light information coming from the entire frame and average giving no weighting to any particular portion of the metered area.
V4L2_EXPOSURE_METERING_CENTER_WEIGHTED	Average the light information coming from the entire frame giving priority to the center of the metered area.
V4L2_EXPOSURE_METERING_SPOT	Measure only very small area at the center of the frame.
V4L2_EXPOSURE_METERING_MATRIX	A multi-zone metering. The light intensity is measured in several points of the frame and the results are combined. The algorithm of the zones selection and their significance in calculating the final value is device dependent.

V4L2_CID_PAN_RELATIVE (integer) This control turns the camera horizontally by the specified amount. The unit is undefined. A positive value moves the camera to the right (clockwise when viewed from above), a negative value to the left. A value of zero does not cause motion. This is a write-only control.

V4L2_CID_TILT_RELATIVE (integer) This control turns the camera vertically by the specified amount. The unit is undefined. A positive value moves the camera up, a negative value down. A value of zero does not cause motion. This is a write-only control.

V4L2_CID_PAN_RESET (button) When this control is set, the camera moves horizontally to the default position.

V4L2_CID_TILT_RESET (button) When this control is set, the camera moves vertically to the default position.

V4L2_CID_PAN_ABSOLUTE (integer) This control turns the camera horizontally to the specified position. Positive values move the camera to the right (clockwise when viewed from above), negative values to the left. Drivers should interpret the values as arc seconds, with valid values between -180 * 3600 and +180 * 3600 inclusive.

V4L2_CID_TILT_ABSOLUTE (integer) This control turns the camera vertically to the specified position. Positive values move the camera up, negative values down. Drivers should interpret the values as arc seconds, with valid values between -180 * 3600 and +180 * 3600 inclusive.

V4L2_CID_FOCUS_ABSOLUTE (integer) This control sets the focal point of the camera to the specified position. The unit is undefined. Positive values set the focus closer to the camera, negative values towards infinity.

V4L2_CID_FOCUS_RELATIVE (integer) This control moves the focal point of the camera by the specified amount. The unit is undefined. Positive values move the focus closer to the camera, negative values towards infinity. This is a write-only control.

V4L2_CID_FOCUS_AUTO (boolean) Enables continuous automatic focus adjustments. The effect of manual focus adjustments while this feature is enabled is undefined, drivers should ignore such requests.

V4L2_CID_AUTO_FOCUS_START (button) Starts single auto focus process. The effect of setting this control when V4L2_CID_FOCUS_AUTO is set to TRUE (1) is undefined, drivers should ignore such requests.

V4L2_CID_AUTO_FOCUS_STOP (button) Aborts automatic focusing started with V4L2_CID_AUTO_FOCUS_START control. It is effective only when the continuous autofocus is disabled, that is when V4L2_CID_FOCUS_AUTO control is set to FALSE (0).

V4L2_CID_AUTO_FOCUS_STATUS (bitmask) The automatic focus status. This is a read-only control.

Setting V4L2_LOCK_FOCUS lock bit of the V4L2_CID_3A_LOCK control may stop updates of the V4L2_CID_AUTO_FOCUS_STATUS control value.

V4L2_AUTO_FOCUS_STATUS_IDLE	Automatic focus is not active.
V4L2_AUTO_FOCUS_STATUS_BUSY	Automatic focusing is in progress.
V4L2_AUTO_FOCUS_STATUS_REACHED	Focus has been reached.
V4L2_AUTO_FOCUS_STATUS_FAILED	Automatic focus has failed, the driver will not transition from this state until another action is performed by an application.

V4L2_CID_AUTO_FOCUS_RANGE (enum)

enum v4l2_auto_focus_range - Determines auto focus distance range for which lens may be adjusted.

V4L2_AUTO_FOCUS_RANGE_AUTO	The camera automatically selects the focus range.
V4L2_AUTO_FOCUS_RANGE_NORMAL	Normal distance range, limited for best automatic focus performance.
V4L2_AUTO_FOCUS_RANGE_MACRO	Macro (close-up) auto focus. The camera will use its minimum possible distance for auto focus.
V4L2_AUTO_FOCUS_RANGE_INFINITY	The lens is set to focus on an object at infinite distance.

V4L2_CID_ZOOM_ABSOLUTE (integer) Specify the objective lens focal length as an absolute value. The zoom unit is driver-specific and its value should be a positive integer.

V4L2_CID_ZOOM_RELATIVE (integer) Specify the objective lens focal length relatively to the current value. Positive values move the zoom lens group towards the telephoto direction, negative values towards the wide-angle direction. The zoom unit is driver-specific. This is a write-only control.

V4L2_CID_ZOOM_CONTINUOUS (integer) Move the objective lens group at the specified speed until it reaches physical device limits or until an explicit request to stop the movement. A positive value moves the zoom lens group towards the telephoto direction. A value of zero stops the zoom lens group movement. A negative value moves the zoom lens group towards the wide-angle direction. The zoom speed unit is driver-specific.

V4L2_CID_IRIS_ABSOLUTE (integer) This control sets the camera's aperture to the specified value. The unit is undefined. Larger values open the iris wider, smaller values close it.

V4L2_CID_IRIS_RELATIVE (integer) This control modifies the camera's aperture by the specified amount. The unit is undefined. Positive values open the iris one step further, negative values close it one step further. This is a write-only control.

V4L2_CID_PRIVACY (boolean) Prevent video from being acquired by the camera. When this control is set to TRUE (1), no image can be captured by the camera. Common means to enforce privacy are mechanical obturation of the sensor and firmware image processing, but the device is not restricted to these methods. Devices that implement the privacy control must support read access and may support write access.

V4L2_CID_BAND_STOP_FILTER (integer) Switch the band-stop filter of a camera sensor on or off, or specify its strength. Such band-stop filters can be used, for example, to filter out the fluorescent light component.

V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE (enum)

enum v4l2_auto_n_preset_white_balance - Sets white balance to automatic, manual or a preset. The presets determine color temperature of the light as a hint to the camera for white balance adjustments resulting in most accurate color representation. The following white balance presets are listed in order of increasing color temperature.

V4L2_WHITE_BALANCE_MANUAL	Manual white balance.
V4L2_WHITE_BALANCE_AUTO	Automatic white balance adjustments.
V4L2_WHITE_BALANCE_INCANDESCENT	White balance setting for incandescent (tungsten) lighting. It generally cools down the colors and corresponds approximately to 2500...3500 K color temperature range.
V4L2_WHITE_BALANCE_FLUORESCENT	White balance preset for fluorescent lighting. It corresponds approximately to 4000...5000 K color temperature.
V4L2_WHITE_BALANCE_FLUORESCENT_H	With this setting the camera will compensate for fluorescent H lighting.
V4L2_WHITE_BALANCE_HORIZON	White balance setting for horizon daylight. It corresponds approximately to 5000 K color temperature.
V4L2_WHITE_BALANCE_DAYLIGHT	White balance preset for daylight (with clear sky). It corresponds approximately to 5000...6500 K color temperature.
V4L2_WHITE_BALANCE_FLASH	With this setting the camera will compensate for the flash light. It slightly warms up the colors and corresponds roughly to 5000...5500 K color temperature.
V4L2_WHITE_BALANCE_CLOUDY	White balance preset for moderately overcast sky. This option corresponds approximately to 6500...8000 K color temperature range.
V4L2_WHITE_BALANCE_SHADE	White balance preset for shade or heavily overcast sky. It corresponds approximately to 9000...10000 K color temperature.

V4L2_CID_WIDE_DYNAMIC_RANGE (boolean) Enables or disables the camera's wide dynamic range feature. This feature allows to obtain clear images in situations where intensity of the illumination varies significantly throughout the scene, i.e. there are simultaneously very dark and very bright areas. It is most commonly realized in cameras by combining two subsequent frames with different exposure times. [1]

V4L2_CID_IMAGE_STABILIZATION (boolean) Enables or disables image stabilization.

V4L2_CID_ISO_SENSITIVITY (integer menu) Determines ISO equivalent of an image sensor indicating the sensor's sensitivity to light. The numbers are expressed in arithmetic scale, as per *ISO 12232:2006* standard, where doubling the sensor sensitivity is represented by doubling the numerical ISO value. Applications should interpret the values as standard ISO values multiplied by 1000, e.g. control value 800 stands for ISO 0.8. Drivers will usually support only a subset of standard ISO values. The effect of setting this control while the V4L2_CID_ISO_SENSITIVITY_AUTO control is set to a value other than V4L2_CID_ISO_SENSITIVITY_MANUAL is undefined, drivers should ignore such requests.

V4L2_CID_ISO_SENSITIVITY_AUTO (enum)

enum v4l2_iso_sensitivity_type - Enables or disables automatic ISO sensitivity adjustments.

V4L2_CID_ISO_SENSITIVITY_MANUAL	Manual ISO sensitivity.
V4L2_CID_ISO_SENSITIVITY_AUTO	Automatic ISO sensitivity adjustments.

V4L2_CID_SCENE_MODE (enum)

enum v4l2_scene_mode - This control allows to select scene programs as the camera automatic modes optimized for common shooting scenes. Within these modes the camera determines best exposure, aperture, focusing, light metering, white balance and equivalent sensitivity. The controls of those parameters are influenced by the scene mode control. An exact behavior in each mode is subject to the camera specification.

When the scene mode feature is not used, this control should be set to V4L2_SCENE_MODE_NONE to make sure the other possibly related controls are accessible. The following scene programs are defined:

[1] This control may be changed to a menu control in the future, if more options are required.

V4L2_SCENE_MODE_NONE	The scene mode feature is disabled.
V4L2_SCENE_MODE_BACKLIGHT	Backlight. Compensates for dark shadows when light is coming from behind a subject, also by automatically turning on the flash.
V4L2_SCENE_MODE_BEACH_SNOW	Beach and snow. This mode compensates for all-white or bright scenes, which tend to look gray and low contrast, when camera's automatic exposure is based on an average scene brightness. To compensate, this mode automatically slightly overexposes the frames. The white balance may also be adjusted to compensate for the fact that reflected snow looks bluish rather than white.
V4L2_SCENE_MODE_CANDLELIGHT	Candle light. The camera generally raises the ISO sensitivity and lowers the shutter speed. This mode compensates for relatively close subject in the scene. The flash is disabled in order to preserve the ambiance of the light.
V4L2_SCENE_MODE_DAWN_DUSK	Dawn and dusk. Preserves the colors seen in low natural light before dusk and after down. The camera may turn off the flash, and automatically focus at infinity. It will usually boost saturation and lower the shutter speed.
V4L2_SCENE_MODE_FALL_COLORS	Fall colors. Increases saturation and adjusts white balance for color enhancement. Pictures of autumn leaves get saturated reds and yellows.
V4L2_SCENE_MODE_FIREWORKS	Fireworks. Long exposure times are used to capture the expanding burst of light from a firework. The camera may invoke image stabilization.
V4L2_SCENE_MODE_LANDSCAPE	Landscape. The camera may choose a small aperture to provide deep depth of field and long exposure duration to help capture detail in dim light conditions. The focus is fixed at infinity. Suitable for distant and wide scenery.
V4L2_SCENE_MODE_NIGHT	Night, also known as Night Landscape. Designed for low light conditions, it preserves detail in the dark areas without blowing out bright objects. The camera generally sets itself to a medium-to-high ISO sensitivity, with a relatively long exposure time, and turns flash off. As such, there will be increased image noise and the possibility of blurred image.
V4L2_SCENE_MODE_PARTY_INDOOR	Party and indoor. Designed to capture indoor scenes that are lit by indoor background lighting as well as the flash. The camera usually increases ISO sensitivity, and adjusts exposure for the low light conditions.
V4L2_SCENE_MODE_PORTRAIT	Portrait. The camera adjusts the aperture so that the depth of field is reduced, which helps to isolate the subject against a smooth background. Most cameras recognize the presence of faces in the scene and focus on them. The color hue is adjusted to enhance skin tones. The intensity of the flash is often reduced.
V4L2_SCENE_MODE_SPORTS	Sports. Significantly increases ISO and uses a fast shutter speed to freeze motion of rapidly-moving subjects. Increased image noise may be seen in this mode.
V4L2_SCENE_MODE_SUNSET	Sunset. Preserves deep hues seen in sunsets and sunrises. It bumps up the saturation.
V4L2_SCENE_MODE_TEXT	Text. It applies extra contrast and sharpness, it is typically a black-and-white mode optimized for readability. Automatic focus may be switched to close-up mode and this setting may also involve some lens-distortion correction.

V4L2_CID_3A_LOCK (bitmask) This control locks or unlocks the automatic focus, exposure and white balance. The automatic adjustments can be paused independently by setting the corresponding lock bit to 1. The camera then retains the settings until the lock bit is cleared. The following lock bits are defined:

When a given algorithm is not enabled, drivers should ignore requests to lock it and should return no error. An example might be an application setting bit V4L2_LOCK_WHITE_BALANCE when the

V4L2_CID_AUTO_WHITE_BALANCE control is set to FALSE. The value of this control may be changed by exposure, white balance or focus controls.

V4L2_LOCK_EXPOSURE	Automatic exposure adjustments lock.
V4L2_LOCK_WHITE_BALANCE	Automatic white balance adjustments lock.
V4L2_LOCK_FOCUS	Automatic focus lock.

V4L2_CID_PAN_SPEED (integer) This control turns the camera horizontally at the specific speed. The unit is undefined. A positive value moves the camera to the right (clockwise when viewed from above), a negative value to the left. A value of zero stops the motion if one is in progress and has no effect otherwise.

V4L2_CID_TILT_SPEED (integer) This control turns the camera vertically at the specified speed. The unit is undefined. A positive value moves the camera up, a negative value down. A value of zero stops the motion if one is in progress and has no effect otherwise.

FM Transmitter Control Reference

The FM Transmitter (FM_TX) class includes controls for common features of FM transmissions capable devices. Currently this class includes parameters for audio compression, pilot tone generation, audio deviation limiter, RDS transmission and tuning power features.

FM_TX Control IDs

V4L2_CID_FM_TX_CLASS (class) The FM_TX class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_RDS_TX_DEVIATION (integer) Configures RDS signal frequency deviation level in Hz. The range and step are driver-specific.

V4L2_CID_RDS_TX_PI (integer) Sets the RDS Programme Identification field for transmission.

V4L2_CID_RDS_TX_PTY (integer) Sets the RDS Programme Type field for transmission. This encodes up to 31 pre-defined programme types.

V4L2_CID_RDS_TX_PS_NAME (string) Sets the Programme Service name (PS_NAME) for transmission. It is intended for static display on a receiver. It is the primary aid to listeners in programme service identification and selection. In Annex E of *IEC 62106* , the RDS specification, there is a full description of the correct character encoding for Programme Service name strings. Also from RDS specification, PS is usually a single eight character text. However, it is also possible to find receivers which can scroll strings sized as 8 x N characters. So, this control must be configured with steps of 8 characters. The result is it must always contain a string with size multiple of 8.

V4L2_CID_RDS_TX_RADIO_TEXT (string) Sets the Radio Text info for transmission. It is a textual description of what is being broadcasted. RDS Radio Text can be applied when broadcaster wishes to transmit longer PS names, programme-related information or any other text. In these cases, RadioText should be used in addition to V4L2_CID_RDS_TX_PS_NAME. The encoding for Radio Text strings is also fully described in Annex E of *IEC 62106* . The length of Radio Text strings depends on which RDS Block is being used to transmit it, either 32 (2A block) or 64 (2B block). However, it is also possible to find receivers which can scroll strings sized as 32 x N or 64 x N characters. So, this control must be configured with steps of 32 or 64 characters. The result is it must always contain a string with size multiple of 32 or 64.

V4L2_CID_RDS_TX_MONO_STEREO (boolean) Sets the Mono/Stereo bit of the Decoder Identification code. If set, then the audio was recorded as stereo.

V4L2_CID_RDS_TX_ARTIFICIAL_HEAD (boolean) Sets the Artificial Head bit of the Decoder Identification code. If set, then the audio was recorded using an artificial head.

V4L2_CID_RDS_TX_COMPRESSED (boolean) Sets the Compressed bit of the Decoder Identification code. If set, then the audio is compressed.

V4L2_CID_RDS_TX_DYNAMIC_PTY (boolean) Sets the Dynamic PTY bit of the Decoder Identification code. If set, then the PTY code is dynamically switched.

V4L2_CID_RDS_TX_TRAFFIC_ANNOUNCEMENT (boolean) If set, then a traffic announcement is in progress.

V4L2_CID_RDS_TX_TRAFFIC_PROGRAM (boolean) If set, then the tuned programme carries traffic announcements.

V4L2_CID_RDS_TX_MUSIC_SPEECH (boolean) If set, then this channel broadcasts music. If cleared, then it broadcasts speech. If the transmitter doesn't make this distinction, then it should be set.

V4L2_CID_RDS_TX_ALT_FREQS_ENABLE (boolean) If set, then transmit alternate frequencies.

V4L2_CID_RDS_TX_ALT_FREQS (__u32 array) The alternate frequencies in kHz units. The RDS standard allows for up to 25 frequencies to be defined. Drivers may support fewer frequencies so check the array size.

V4L2_CID_AUDIO_LIMITER_ENABLED (boolean) Enables or disables the audio deviation limiter feature. The limiter is useful when trying to maximize the audio volume, minimize receiver-generated distortion and prevent overmodulation.

V4L2_CID_AUDIO_LIMITER_RELEASE_TIME (integer) Sets the audio deviation limiter feature release time. Unit is in useconds. Step and range are driver-specific.

V4L2_CID_AUDIO_LIMITER_DEVIATION (integer) Configures audio frequency deviation level in Hz. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_ENABLED (boolean) Enables or disables the audio compression feature. This feature amplifies signals below the threshold by a fixed gain and compresses audio signals above the threshold by the ratio of Threshold/(Gain + Threshold).

V4L2_CID_AUDIO_COMPRESSION_GAIN (integer) Sets the gain for audio compression feature. It is a dB value. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_THRESHOLD (integer) Sets the threshold level for audio compression freature. It is a dB value. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_ATTACK_TIME (integer) Sets the attack time for audio compression feature. It is a useconds value. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_RELEASE_TIME (integer) Sets the release time for audio compression feature. It is a useconds value. The range and step are driver-specific.

V4L2_CID_PILOT_TONE_ENABLED (boolean) Enables or disables the pilot tone generation feature.

V4L2_CID_PILOT_TONE_DEVIATION (integer) Configures pilot tone frequency deviation level. Unit is in Hz. The range and step are driver-specific.

V4L2_CID_PILOT_TONE_FREQUENCY (integer) Configures pilot tone frequency value. Unit is in Hz. The range and step are driver-specific.

V4L2_CID_TUNE_PREEMPHASIS (enum)

enum v4l2_preemphasis - Configures the pre-emphasis value for broadcasting. A pre-emphasis filter is applied to the broadcast to accentuate the high audio frequencies. Depending on the region, a time constant of either 50 or 75 useconds is used. The enum v4l2_preemphasis defines possible values for pre-emphasis. Here they are:

V4L2_PREEMPHASIS_DISABLED	No pre-emphasis is applied.
V4L2_PREEMPHASIS_50_uS	A pre-emphasis of 50 uS is used.
V4L2_PREEMPHASIS_75_uS	A pre-emphasis of 75 uS is used.

V4L2_CID_TUNE_POWER_LEVEL (integer) Sets the output power level for signal transmission. Unit is in dBuV. Range and step are driver-specific.

V4L2_CID_TUNE_ANTENNA_CAPACITOR (integer) This selects the value of antenna tuning capacitor manually or automatically if set to zero. Unit, range and step are driver-specific.

For more details about RDS specification, refer to *IEC 62106* document, from CENELEC.

Flash Control Reference

The V4L2 flash controls are intended to provide generic access to flash controller devices. Flash controller devices are typically used in digital cameras.

The interface can support both LED and xenon flash devices. As of writing this, there is no xenon flash driver using this interface.

Supported use cases

Unsynchronised LED flash (software strobe)

Unsynchronised LED flash is controlled directly by the host as the sensor. The flash must be enabled by the host before the exposure of the image starts and disabled once it ends. The host is fully responsible for the timing of the flash.

Example of such device: Nokia N900.

Synchronised LED flash (hardware strobe)

The synchronised LED flash is pre-programmed by the host (power and timeout) but controlled by the sensor through a strobe signal from the sensor to the flash.

The sensor controls the flash duration and timing. This information typically must be made available to the sensor.

LED flash as torch

LED flash may be used as torch in conjunction with another use case involving camera or individually.

Flash Control IDs

V4L2_CID_FLASH_CLASS (class) The FLASH class descriptor.

V4L2_CID_FLASH_LED_MODE (menu) Defines the mode of the flash LED, the high-power white LED attached to the flash controller. Setting this control may not be possible in presence of some faults. See V4L2_CID_FLASH_FAULT.

V4L2_FLASH_LED_MODE_NONE	Off.
V4L2_FLASH_LED_MODE_FLASH	Flash mode.
V4L2_FLASH_LED_MODE_TORCH	Torch mode. See V4L2_CID_FLASH_TORCH_INTENSITY.

V4L2_CID_FLASH_STROBE_SOURCE (menu) Defines the source of the flash LED strobe.

V4L2_FLASH_STROBE_SOURCE_SOFTWARE	The flash strobe is triggered by using the V4L2_CID_FLASH_STROBE control.
V4L2_FLASH_STROBE_SOURCE_EXTERNAL	The flash strobe is triggered by an external source. Typically this is a sensor, which makes it possible to synchronises the flash strobe start to exposure start.

V4L2_CID_FLASH_STROBE (button) Strobe flash. Valid when V4L2_CID_FLASH_LED_MODE is set to V4L2_FLASH_LED_MODE_FLASH and V4L2_CID_FLASH_STROBE_SOURCE is set to V4L2_FLASH_STROBE_SOURCE_SOFTWARE. Setting this control may not be possible in presence of some faults. See V4L2_CID_FLASH_FAULT.

V4L2_CID_FLASH_STROBE_STOP (button) Stop flash strobe immediately.

V4L2_CID_FLASH_STROBE_STATUS (boolean) Strobe status: whether the flash is strobing at the moment or not. This is a read-only control.

V4L2_CID_FLASH_TIMEOUT (integer) Hardware timeout for flash. The flash strobe is stopped after this period of time has passed from the start of the strobe.

V4L2_CID_FLASH_INTENSITY (integer) Intensity of the flash strobe when the flash LED is in flash mode (V4L2_FLASH_LED_MODE_FLASH). The unit should be milliamps (mA) if possible.

V4L2_CID_FLASH_TORCH_INTENSITY (integer) Intensity of the flash LED in torch mode (V4L2_FLASH_LED_MODE_TORCH). The unit should be milliamps (mA) if possible. Setting this control may not be possible in presence of some faults. See V4L2_CID_FLASH_FAULT.

V4L2_CID_FLASH_INDICATOR_INTENSITY (integer) Intensity of the indicator LED. The indicator LED may be fully independent of the flash LED. The unit should be microamps (uA) if possible.

V4L2_CID_FLASH_FAULT (bitmask) Faults related to the flash. The faults tell about specific problems in the flash chip itself or the LEDs attached to it. Faults may prevent further use of some of the flash controls. In particular, V4L2_CID_FLASH_LED_MODE is set to V4L2_FLASH_LED_MODE_NONE if the fault affects the flash LED. Exactly which faults have such an effect is chip dependent. Reading the faults resets the control and returns the chip to a usable state if possible.

V4L2_FLASH_FAULT_OVER_VOLTAGE	Flash controller voltage to the flash LED has exceeded the limit specific to the flash controller.
V4L2_FLASH_FAULT_TIMEOUT	The flash strobe was still on when the timeout set by the user — V4L2_CID_FLASH_TIMEOUT control — has expired. Not all flash controllers may set this in all such conditions.
V4L2_FLASH_FAULT_OVER_TEMPERATURE	The flash controller has overheated.
V4L2_FLASH_FAULT_SHORT_CIRCUIT	The short circuit protection of the flash controller has been triggered.
V4L2_FLASH_FAULT_OVER_CURRENT	Current in the LED power supply has exceeded the limit specific to the flash controller.
V4L2_FLASH_FAULT_INDICATOR	The flash controller has detected a short or open circuit condition on the indicator LED.
V4L2_FLASH_FAULT_UNDER_VOLTAGE	Flash controller voltage to the flash LED has been below the minimum limit specific to the flash controller.
V4L2_FLASH_FAULT_INPUT_VOLTAGE	The input voltage of the flash controller is below the limit under which strobing the flash at full current will not be possible.The condition persists until this flag is no longer set.
V4L2_FLASH_FAULT_LED_OVER_TEMPERATURE	The temperature of the LED has exceeded its allowed upper limit.

V4L2_CID_FLASH_CHARGE (boolean) Enable or disable charging of the xenon flash capacitor.

V4L2_CID_FLASH_READY (boolean) Is the flash ready to strobe? Xenon flashes require their capacitors charged before strobing. LED flashes often require a cooldown period after strobe during which another strobe will not be possible. This is a read-only control.

JPEG Control Reference

The JPEG class includes controls for common features of JPEG encoders and decoders. Currently it includes features for codecs implementing progressive baseline DCT compression process with Huffman entrophy coding.

JPEG Control IDs

V4L2_CID_JPEG_CLASS (class) The JPEG class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this con-

trol class.

V4L2_CID_JPEG_CHROMA_SUBSAMPLING (menu) The chroma subsampling factors describe how each component of an input image is sampled, in respect to maximum sample rate in each spatial dimension. See *ITU-T.81* , clause A.1.1. for more details. The V4L2_CID_JPEG_CHROMA_SUBSAMPLING control determines how Cb and Cr components are downsampled after converting an input image from RGB to Y'CbCr color space.

V4L2_JPEG_CHROMA_SUBSAMPLING_444	No chroma subsampling, each pixel has Y, Cr and Cb values.
V4L2_JPEG_CHROMA_SUBSAMPLING_422	Horizontally subsample Cr, Cb components by a factor of 2.
V4L2_JPEG_CHROMA_SUBSAMPLING_420	Subsample Cr, Cb components horizontally and vertically by 2.
V4L2_JPEG_CHROMA_SUBSAMPLING_411	Horizontally subsample Cr, Cb components by a factor of 4.
V4L2_JPEG_CHROMA_SUBSAMPLING_410	Subsample Cr, Cb components horizontally by 4 and vertically by 2.
V4L2_JPEG_CHROMA_SUBSAMPLING_GRAY	Use only luminance component.

V4L2_CID_JPEG_RESTART_INTERVAL (integer) The restart interval determines an interval of inserting RSTm markers (m = 0..7). The purpose of these markers is to additionally reinitialize the encoder process, in order to process blocks of an image independently. For the lossy compression processes the restart interval unit is MCU (Minimum Coded Unit) and its value is contained in DRI (Define Restart Interval) marker. If V4L2_CID_JPEG_RESTART_INTERVAL control is set to 0, DRI and RSTm markers will not be inserted.

V4L2_CID_JPEG_COMPRESSION_QUALITY (integer) V4L2_CID_JPEG_COMPRESSION_QUALITY control determines trade-off between image quality and size. It provides simpler method for applications to control image quality, without a need for direct reconfiguration of luminance and chrominance quantization tables. In cases where a driver uses quantization tables configured directly by an application, using interfaces defined elsewhere, V4L2_CID_JPEG_COMPRESSION_QUALITY control should be set by driver to 0.

The value range of this control is driver-specific. Only positive, non-zero values are meaningful. The recommended range is 1 - 100, where larger values correspond to better image quality.

V4L2_CID_JPEG_ACTIVE_MARKER (bitmask) Specify which JPEG markers are included in compressed stream. This control is valid only for encoders.

V4L2_JPEG_ACTIVE_MARKER_APP0	Application data segment APP_0.
V4L2_JPEG_ACTIVE_MARKER_APP1	Application data segment APP_1.
V4L2_JPEG_ACTIVE_MARKER_COM	Comment segment.
V4L2_JPEG_ACTIVE_MARKER_DQT	Quantization tables segment.
V4L2_JPEG_ACTIVE_MARKER_DHT	Huffman tables segment.

For more details about JPEG specification, refer to *ITU-T.81* , *JFIF* , *W3C JPEG JFIF* .

Image Source Control Reference

The Image Source control class is intended for low-level control of image source devices such as image sensors. The devices feature an analogue to digital converter and a bus transmitter to transmit the image data out of the device.

Image Source Control IDs

V4L2_CID_IMAGE_SOURCE_CLASS (class) The IMAGE_SOURCE class descriptor.

V4L2_CID_VBLANK (integer) Vertical blanking. The idle period after every frame during which no image data is produced. The unit of vertical blanking is a line. Every line has length of the image width plus horizontal blanking at the pixel rate defined by V4L2_CID_PIXEL_RATE control in the same subdevice.

V4L2_CID_HBLANK (integer) Horizontal blanking. The idle period after every line of image data during which no image data is produced. The unit of horizontal blanking is pixels.

V4L2_CID_ANALOGUE_GAIN (integer) Analogue gain is gain affecting all colour components in the pixel matrix. The gain operation is performed in the analogue domain before A/D conversion.

V4L2_CID_TEST_PATTERN_RED (integer) Test pattern red colour component.

V4L2_CID_TEST_PATTERN_GREENR (integer) Test pattern green (next to red) colour component.

V4L2_CID_TEST_PATTERN_BLUE (integer) Test pattern blue colour component.

V4L2_CID_TEST_PATTERN_GREENB (integer) Test pattern green (next to blue) colour component.

Image Process Control Reference

The Image Process control class is intended for low-level control of image processing functions. Unlike V4L2_CID_IMAGE_SOURCE_CLASS, the controls in this class affect processing the image, and do not control capturing of it.

Image Process Control IDs

V4L2_CID_IMAGE_PROC_CLASS (class) The IMAGE_PROC class descriptor.

V4L2_CID_LINK_FREQ (integer menu) Data bus frequency. Together with the media bus pixel code, bus type (clock cycles per sample), the data bus frequency defines the pixel rate (V4L2_CID_PIXEL_RATE) in the pixel array (or possibly elsewhere, if the device is not an image sensor). The frame rate can be calculated from the pixel clock, image width and height and horizontal and vertical blanking. While the pixel rate control may be defined elsewhere than in the subdev containing the pixel array, the frame rate cannot be obtained from that information. This is because only on the pixel array it can be assumed that the vertical and horizontal blanking information is exact: no other blanking is allowed in the pixel array. The selection of frame rate is performed by selecting the desired horizontal and vertical blanking. The unit of this control is Hz.

V4L2_CID_PIXEL_RATE (64-bit integer) Pixel rate in the source pads of the subdev. This control is read-only and its unit is pixels / second.

V4L2_CID_TEST_PATTERN (menu) Some capture/display/sensor devices have the capability to generate test pattern images. These hardware specific test patterns can be used to test if a device is working properly.

V4L2_CID_DEINTERLACING_MODE (menu) The video deinterlacing mode (such as Bob, Weave, ...). The menu items are driver specific and are documented in *Video4Linux (V4L) driver-specific documentation* .

Digital Video Control Reference

The Digital Video control class is intended to control receivers and transmitters for VGA, DVI (Digital Visual Interface), HDMI (*HDMI*) and DisplayPort (*DP*). These controls are generally expected to be private to the receiver or transmitter subdevice that implements them, so they are only exposed on the /dev/v4l-subdev* device node.

Note:

Note that these devices can have multiple input or output pads which are hooked up to e.g. HDMI connectors. Even though the subdevice will receive or transmit video from/to only one of those pads, the other pads can still be active when it comes to EDID (Extended Display Identification Data, EDID *) and HDCP (High-bandwidth Digital Content Protection System,* HDCP *) processing, allowing the device to do the fairly slow EDID/HDCP handling in advance. This allows for quick switching between connectors.*

These pads appear in several of the controls in this section as bitmasks, one bit for each pad. Bit 0 corresponds to pad 0, bit 1 to pad 1, etc. The maximum value of the control is the set of valid pads.

Digital Video Control IDs

V4L2_CID_DV_CLASS (class) The Digital Video class descriptor.

V4L2_CID_DV_TX_HOTPLUG (bitmask) Many connectors have a hotplug pin which is high if EDID information is available from the source. This control shows the state of the hotplug pin as seen by the transmitter. Each bit corresponds to an output pad on the transmitter. If an output pad does not have an associated hotplug pin, then the bit for that pad will be 0. This read-only control is applicable to DVI-D, HDMI and DisplayPort connectors.

V4L2_CID_DV_TX_RXSENSE (bitmask) Rx Sense is the detection of pull-ups on the TMDS clock lines. This normally means that the sink has left/entered standby (i.e. the transmitter can sense that the receiver is ready to receive video). Each bit corresponds to an output pad on the transmitter. If an output pad does not have an associated Rx Sense, then the bit for that pad will be 0. This read-only control is applicable to DVI-D and HDMI devices.

V4L2_CID_DV_TX_EDID_PRESENT (bitmask) When the transmitter sees the hotplug signal from the receiver it will attempt to read the EDID. If set, then the transmitter has read at least the first block (= 128 bytes). Each bit corresponds to an output pad on the transmitter. If an output pad does not support EDIDs, then the bit for that pad will be 0. This read-only control is applicable to VGA, DVI-A/D, HDMI and DisplayPort connectors.

V4L2_CID_DV_TX_MODE (enum)

enum v4l2_dv_tx_mode - HDMI transmitters can transmit in DVI-D mode (just video) or in HDMI mode (video + audio + auxiliary data). This control selects which mode to use: V4L2_DV_TX_MODE_DVI_D or V4L2_DV_TX_MODE_HDMI. This control is applicable to HDMI connectors.

V4L2_CID_DV_TX_RGB_RANGE (enum)

enum v4l2_dv_rgb_range - Select the quantization range for RGB output. V4L2_DV_RANGE_AUTO follows the RGB quantization range specified in the standard for the video interface (ie. *CEA-861-E* for HDMI). V4L2_DV_RANGE_LIMITED and V4L2_DV_RANGE_FULL override the standard to be compatible with sinks that have not implemented the standard correctly (unfortunately quite common for HDMI and DVI-D). Full range allows all possible values to be used whereas limited range sets the range to (16 << (N-8)) - (235 << (N-8)) where N is the number of bits per component. This control is applicable to VGA, DVI-A/D, HDMI and DisplayPort connectors.

V4L2_CID_DV_TX_IT_CONTENT_TYPE (enum)

enum v4l2_dv_it_content_type - Configures the IT Content Type of the transmitted video. This information is sent over HDMI and DisplayPort connectors as part of the AVI InfoFrame. The term 'IT Content' is used for content that originates from a computer as opposed to content from a TV broadcast or an analog source. The enum v4l2_dv_it_content_type defines the possible content types:

V4L2_DV_IT_CONTENT_TYPE_GRAPHICS	Graphics content. Pixel data should be passed unfiltered and without analog reconstruction.
V4L2_DV_IT_CONTENT_TYPE_PHOTO	Photo content. The content is derived from digital still pictures. The content should be passed through with minimal scaling and picture enhancements.
V4L2_DV_IT_CONTENT_TYPE_CINEMA	Cinema content.
V4L2_DV_IT_CONTENT_TYPE_GAME	Game content. Audio and video latency should be minimized.
V4L2_DV_IT_CONTENT_TYPE_NO_ITC	No IT Content information is available and the ITC bit in the AVI InfoFrame is set to 0.

V4L2_CID_DV_RX_POWER_PRESENT (bitmask) Detects whether the receiver receives power from the source (e.g. HDMI carries 5V on one of the pins). This is often used to power an eeprom which contains EDID information, such that the source can read the EDID even if the sink is in standby/power off. Each bit corresponds to an input pad on the transmitter. If an input pad cannot detect whether power is present, then the bit for that pad will be 0. This read-only control is applicable to DVI-D, HDMI and DisplayPort connectors.

V4L2_CID_DV_RX_RGB_RANGE (enum)

enum v4l2_dv_rgb_range - Select the quantization range for RGB input. V4L2_DV_RANGE_AUTO follows the RGB quantization range specified in the standard for the video interface (ie. *CEA-861-E* for HDMI). V4L2_DV_RANGE_LIMITED and V4L2_DV_RANGE_FULL override the standard to be compatible with sources that have not implemented the standard correctly (unfortunately quite common for HDMI and DVI-D). Full range allows all possible values to be used whereas limited range sets the range to (16 << (N-8)) - (235 << (N-8)) where N is the number of bits per component. This control is applicable to VGA, DVI-A/D, HDMI and DisplayPort connectors.

V4L2_CID_DV_RX_IT_CONTENT_TYPE (enum)

enum v4l2_dv_it_content_type - Reads the IT Content Type of the received video. This information is sent over HDMI and DisplayPort connectors as part of the AVI InfoFrame. The term 'IT Content' is used for content that originates from a computer as opposed to content from a TV broadcast or an analog source. See V4L2_CID_DV_TX_IT_CONTENT_TYPE for the available content types.

FM Receiver Control Reference

The FM Receiver (FM_RX) class includes controls for common features of FM Reception capable devices.

FM_RX Control IDs

V4L2_CID_FM_RX_CLASS (class) The FM_RX class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_RDS_RECEPTION (boolean) Enables/disables RDS reception by the radio tuner

V4L2_CID_RDS_RX_PTY (integer) Gets RDS Programme Type field. This encodes up to 31 pre-defined programme types.

V4L2_CID_RDS_RX_PS_NAME (string) Gets the Programme Service name (PS_NAME). It is intended for static display on a receiver. It is the primary aid to listeners in programme service identification and selection. In Annex E of *IEC 62106* , the RDS specification, there is a full description of the correct character encoding for Programme Service name strings. Also from RDS specification, PS is usually a single eight character text. However, it is also possible to find receivers which can scroll strings sized as 8 x N characters. So, this control must be configured with steps of 8 characters. The result is it must always contain a string with size multiple of 8.

V4L2_CID_RDS_RX_RADIO_TEXT (string) Gets the Radio Text info. It is a textual description of what is being broadcasted. RDS Radio Text can be applied when broadcaster wishes to transmit longer PS

names, programme-related information or any other text. In these cases, RadioText can be used in addition to V4L2_CID_RDS_RX_PS_NAME. The encoding for Radio Text strings is also fully described in Annex E of *IEC 62106* . The length of Radio Text strings depends on which RDS Block is being used to transmit it, either 32 (2A block) or 64 (2B block). However, it is also possible to find receivers which can scroll strings sized as 32 x N or 64 x N characters. So, this control must be configured with steps of 32 or 64 characters. The result is it must always contain a string with size multiple of 32 or 64.

V4L2_CID_RDS_RX_TRAFFIC_ANNOUNCEMENT (boolean) If set, then a traffic announcement is in progress.

V4L2_CID_RDS_RX_TRAFFIC_PROGRAM (boolean) If set, then the tuned programme carries traffic announcements.

V4L2_CID_RDS_RX_MUSIC_SPEECH (boolean) If set, then this channel broadcasts music. If cleared, then it broadcasts speech. If the transmitter doesn't make this distinction, then it will be set.

V4L2_CID_TUNE_DEEMPHASIS (enum)

enum v4l2_deemphasis - Configures the de-emphasis value for reception. A de-emphasis filter is applied to the broadcast to accentuate the high audio frequencies. Depending on the region, a time constant of either 50 or 75 useconds is used. The enum v4l2_deemphasis defines possible values for de-emphasis. Here they are:

V4L2_DEEMPHASIS_DISABLED	No de-emphasis is applied.
V4L2_DEEMPHASIS_50_uS	A de-emphasis of 50 uS is used.
V4L2_DEEMPHASIS_75_uS	A de-emphasis of 75 uS is used.

Detect Control Reference

The Detect class includes controls for common features of various motion or object detection capable devices.

Detect Control IDs

V4L2_CID_DETECT_CLASS (class) The Detect class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_DETECT_MD_MODE (menu) Sets the motion detection mode.

V4L2_DETECT_MD_MODE_DISABLED	Disable motion detection.
V4L2_DETECT_MD_MODE_GLOBAL	Use a single motion detection threshold.
V4L2_DETECT_MD_MODE_THRESHOLD_GRID	The image is divided into a grid, each cell with its own motion detection threshold. These thresholds are set through the V4L2_CID_DETECT_MD_THRESHOLD_GRID matrix control.
V4L2_DETECT_MD_MODE_REGION_GRID	The image is divided into a grid, each cell with its own region value that specifies which per-region motion detection thresholds should be used. Each region has its own thresholds. How these per-region thresholds are set up is driver-specific. The region values for the grid are set through the V4L2_CID_DETECT_MD_REGION_GRID matrix control.

V4L2_CID_DETECT_MD_GLOBAL_THRESHOLD (integer) Sets the global motion detection threshold to be used with the V4L2_DETECT_MD_MODE_GLOBAL motion detection mode.

V4L2_CID_DETECT_MD_THRESHOLD_GRID (__u16 matrix) Sets the motion detection thresholds for each cell in the grid. To be used with the V4L2_DETECT_MD_MODE_THRESHOLD_GRID motion detection mode. Matrix element (0, 0) represents the cell at the top-left of the grid.

V4L2_CID_DETECT_MD_REGION_GRID (__u8 matrix) Sets the motion detection region value for each cell in the grid. To be used with the V4L2_DETECT_MD_MODE_REGION_GRID motion detection mode. Matrix element (0, 0) represents the cell at the top-left of the grid.

RF Tuner Control Reference

The RF Tuner (RF_TUNER) class includes controls for common features of devices having RF tuner.

In this context, RF tuner is radio receiver circuit between antenna and demodulator. It receives radio frequency (RF) from the antenna and converts that received signal to lower intermediate frequency (IF) or baseband frequency (BB). Tuners that could do baseband output are often called Zero-IF tuners. Older tuners were typically simple PLL tuners inside a metal box, whilst newer ones are highly integrated chips without a metal box "silicon tuners". These controls are mostly applicable for new feature rich silicon tuners, just because older tuners does not have much adjustable features.

For more information about RF tuners see Tuner (radio) and RF front end from Wikipedia.

RF_TUNER Control IDs

V4L2_CID_RF_TUNER_CLASS (class) The RF_TUNER class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_RF_TUNER_BANDWIDTH_AUTO (boolean) Enables/disables tuner radio channel bandwidth configuration. In automatic mode bandwidth configuration is performed by the driver.

V4L2_CID_RF_TUNER_BANDWIDTH (integer) Filter(s) on tuner signal path are used to filter signal according to receiving party needs. Driver configures filters to fulfill desired bandwidth requirement. Used when V4L2_CID_RF_TUNER_BANDWIDTH_AUTO is not set. Unit is in Hz. The range and step are driver-specific.

V4L2_CID_RF_TUNER_LNA_GAIN_AUTO (boolean) Enables/disables LNA automatic gain control (AGC)

V4L2_CID_RF_TUNER_MIXER_GAIN_AUTO (boolean) Enables/disables mixer automatic gain control (AGC)

V4L2_CID_RF_TUNER_IF_GAIN_AUTO (boolean) Enables/disables IF automatic gain control (AGC)

V4L2_CID_RF_TUNER_RF_GAIN (integer) The RF amplifier is the very first amplifier on the receiver signal path, just right after the antenna input. The difference between the LNA gain and the RF gain in this document is that the LNA gain is integrated in the tuner chip while the RF gain is a separate chip. There may be both RF and LNA gain controls in the same device. The range and step are driver-specific.

V4L2_CID_RF_TUNER_LNA_GAIN (integer) LNA (low noise amplifier) gain is first gain stage on the RF tuner signal path. It is located very close to tuner antenna input. Used when V4L2_CID_RF_TUNER_LNA_GAIN_AUTO is not set. See V4L2_CID_RF_TUNER_RF_GAIN to understand how RF gain and LNA gain differs from the each others. The range and step are driver-specific.

V4L2_CID_RF_TUNER_MIXER_GAIN (integer) Mixer gain is second gain stage on the RF tuner signal path. It is located inside mixer block, where RF signal is down-converted by the mixer. Used when V4L2_CID_RF_TUNER_MIXER_GAIN_AUTO is not set. The range and step are driver-specific.

V4L2_CID_RF_TUNER_IF_GAIN (integer) IF gain is last gain stage on the RF tuner signal path. It is located on output of RF tuner. It controls signal level of intermediate frequency output or baseband output. Used when V4L2_CID_RF_TUNER_IF_GAIN_AUTO is not set. The range and step are driver-specific.

V4L2_CID_RF_TUNER_PLL_LOCK (boolean) Is synthesizer PLL locked? RF tuner is receiving given frequency when that control is set. This is a read-only control.

Data Formats

Data Format Negotiation

Different devices exchange different kinds of data with applications, for example video images, raw or sliced VBI data, RDS datagrams. Even within one kind many different formats are possible, in particular an abundance of image formats. Although drivers must provide a default and the selection persists across closing and reopening a device, applications should always negotiate a data format before engaging in data exchange. Negotiation means the application asks for a particular format and the driver selects and reports the best the hardware can do to satisfy the request. Of course applications can also just query the current selection.

A single mechanism exists to negotiate all data formats using the aggregate struct *v4l2_format* and the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls. Additionally the *VIDIOC_TRY_FMT* ioctl can be used to examine what the hardware *could* do, without actually selecting a new data format. The data formats supported by the V4L2 API are covered in the respective device section in *Interfaces* . For a closer look at image formats see *Image Formats* .

The *VIDIOC_S_FMT* ioctl is a major turning-point in the initialization sequence. Prior to this point multiple panel applications can access the same device concurrently to select the current input, change controls or modify other properties. The first *VIDIOC_S_FMT* assigns a logical stream (video data, VBI data etc.) exclusively to one file descriptor.

Exclusive means no other application, more precisely no other file descriptor, can grab this stream or change device properties inconsistent with the negotiated parameters. A video standard change for example, when the new standard uses a different number of scan lines, can invalidate the selected image format. Therefore only the file descriptor owning the stream can make invalidating changes. Accordingly multiple file descriptors which grabbed different logical streams prevent each other from interfering with their settings. When for example video overlay is about to start or already in progress, simultaneous video capturing may be restricted to the same cropping and image size.

When applications omit the *VIDIOC_S_FMT* ioctl its locking side effects are implied by the next step, the selection of an I/O method with the *ioctl VIDIOC_REQBUFS* ioctl or implicit with the first *read()* or *write()* call.

Generally only one logical stream can be assigned to a file descriptor, the exception being drivers permitting simultaneous video capturing and overlay using the same file descriptor for compatibility with V4L and earlier versions of V4L2. Switching the logical stream or returning into “panel mode” is possible by closing and reopening the device. Drivers *may* support a switch using *VIDIOC_S_FMT* .

All drivers exchanging data with applications must support the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl. Implementation of the *VIDIOC_TRY_FMT* is highly recommended but optional.

Image Format Enumeration

Apart of the generic format negotiation functions a special ioctl to enumerate all image formats supported by video capture, overlay or output devices is available. [1]

The *ioctl VIDIOC_ENUM_FMT* ioctl must be supported by all drivers exchanging image data with applications.

Important

Drivers are not supposed to convert image formats in kernel space. They must enumerate only formats directly supported by the hardware. If necessary driver writers should publish an example conversion routine or library for integration into applications.

[1] Enumerating formats an application has no a-priori knowledge of (otherwise it could explicitly ask for them and need not enumerate) seems useless, but there are applications serving as proxy between drivers and the actual video applications for which this is useful.

Single- and multi-planar APIs

Some devices require data for each input or output video frame to be placed in discontiguous memory buffers. In such cases, one video frame has to be addressed using more than one memory address, i.e. one pointer per “plane”. A plane is a sub-buffer of the current frame. For examples of such formats see *Image Formats* .

Initially, V4L2 API did not support multi-planar buffers and a set of extensions has been introduced to handle them. Those extensions constitute what is being referred to as the “multi-planar API”.

Some of the V4L2 API calls and structures are interpreted differently, depending on whether single- or multi-planar API is being used. An application can choose whether to use one or the other by passing a corresponding buffer type to its ioctl calls. Multi-planar versions of buffer types are suffixed with an `_MPLANE` string. For a list of available multi-planar buffer types see enum *v4l2_buf_type*.

Multi-planar formats

Multi-planar API introduces new multi-planar formats. Those formats use a separate set of FourCC codes. It is important to distinguish between the multi-planar API and a multi-planar format. Multi-planar API calls can handle all single-planar formats as well (as long as they are passed in multi-planar API structures), while the single-planar API cannot handle multi-planar formats.

Calls that distinguish between single and multi-planar APIs

VIDIOC_QUERYCAP Two additional multi-planar capabilities are added. They can be set together with non-multi-planar ones for devices that handle both single- and multi-planar formats.

VIDIOC_G_FMT* , *VIDIOC_S_FMT* , *VIDIOC_TRY_FMT New structures for describing multi-planar formats are added: struct *v4l2_pix_format_mplane* and struct *v4l2_plane_pix_format*. Drivers may define new multi-planar formats, which have distinct FourCC codes from the existing single-planar ones.

VIDIOC_QBUF* , *VIDIOC_DQBUF* , *VIDIOC_QUERYBUF A new struct *v4l2_plane* structure for describing planes is added. Arrays of this structure are passed in the new `m.planes` field of struct *v4l2_buffer*.

VIDIOC_REQBUFS Will allocate multi-planar buffers as requested.

Image Cropping, Insertion and Scaling

Some video capture devices can sample a subsection of the picture and shrink or enlarge it to an image of arbitrary size. We call these abilities cropping and scaling. Some video output devices can scale an image up or down and insert it at an arbitrary scan line and horizontal offset into a video signal.

Applications can use the following API to select an area in the video signal, query the default area and the hardware limits.

Note:

Despite their name, the VIDIOC_CROPCAP *,* VIDIOC_G_CROP *and* VIDIOC_S_CROP *ioctls apply to input as well as output devices.*

Scaling requires a source and a target. On a video capture or overlay device the source is the video signal, and the cropping ioctls determine the area actually sampled. The target are images read by the application or overlaid onto the graphics screen. Their size (and position for an overlay) is negotiated with the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls.

On a video output device the source are the images passed in by the application, and their size is again negotiated with the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls, or may be encoded in a compressed video stream. The target is the video signal, and the cropping ioctls determine the area where the images are inserted.

Source and target rectangles are defined even if the device does not support scaling or the *VIDIOC_G_CROP* and *VIDIOC_S_CROP* ioctls. Their size (and position where applicable) will be fixed in this case.

Note:

All capture and output devices must support the VIDIOC_CROPCAP *ioctl such that applications can determine if scaling takes place.*

Cropping Structures

Fig. 1.2: Image Cropping, Insertion and Scaling
The cropping, insertion and scaling process

For capture devices the coordinates of the top left corner, width and height of the area which can be sampled is given by the bounds substructure of the struct *v4l2_cropcap* returned by the *VIDIOC_CROPCAP* ioctl. To support a wide range of hardware this specification does not define an origin or units. However by convention drivers should horizontally count unscaled samples relative to 0H (the leading edge of the horizontal sync pulse, see *Figure 4.1. Line synchronization*). Vertically ITU-R line numbers of the first field (see ITU R-525 line numbering for *525 lines* and for *625 lines*), multiplied by two if the driver can capture both fields.

The top left corner, width and height of the source rectangle, that is the area actually sampled, is given by struct *v4l2_crop* using the same coordinate system as struct *v4l2_cropcap*. Applications can use the *VIDIOC_G_CROP* and *VIDIOC_S_CROP* ioctls to get and set this rectangle. It must lie completely within the capture boundaries and the driver may further adjust the requested size and/or position according to hardware limitations.

Each capture device has a default source rectangle, given by the `defrect` substructure of struct *v4l2_cropcap*. The center of this rectangle shall align with the center of the active picture area of the video signal, and cover what the driver writer considers the complete picture. Drivers shall reset the source rectangle to the default when the driver is first loaded, but not later.

For output devices these structures and ioctls are used accordingly, defining the *target* rectangle where the images will be inserted into the video signal.

Scaling Adjustments

Video hardware can have various cropping, insertion and scaling limitations. It may only scale up or down, support only discrete scaling factors, or have different scaling abilities in horizontal and vertical direction. Also it may not support scaling at all. At the same time the struct *v4l2_crop* rectangle may have to be aligned, and both the source and target rectangles may have arbitrary upper and lower size limits. In particular the maximum `width` and `height` in struct *v4l2_crop* may be smaller than the struct *v4l2_cropcap*. bounds area. Therefore, as usual, drivers are expected to adjust the requested parameters and return the actual values selected.

Applications can change the source or the target rectangle first, as they may prefer a particular image size or a certain area in the video signal. If the driver has to adjust both to satisfy hardware limitations, the last requested rectangle shall take priority, and the driver should preferably adjust the opposite one. The *VIDIOC_TRY_FMT* ioctl however shall not change the driver state and therefore only adjust the requested rectangle.

Suppose scaling on a video capture device is restricted to a factor 1:1 or 2:1 in either direction and the target image size must be a multiple of 16 × 16 pixels. The source cropping rectangle is set to defaults, which are also the upper limit in this example, of 640 × 400 pixels at offset 0, 0. An application requests an image size of 300 × 225 pixels, assuming video will be scaled down from the "full picture" accordingly. The driver sets the image size to the closest possible values 304 × 224, then chooses the cropping rectangle closest to the requested size, that is 608 × 224 (224 × 2:1 would exceed the limit 400). The offset 0, 0 is still valid, thus unmodified. Given the default cropping rectangle reported by *VIDIOC_CROPCAP* the application can easily propose another offset to center the cropping rectangle.

Now the application may insist on covering an area using a picture aspect ratio closer to the original request, so it asks for a cropping rectangle of 608 × 456 pixels. The present scaling factors limit cropping to 640 × 384, so the driver returns the cropping size 608 × 384 and adjusts the image size to closest possible 304 × 192.

Examples

Source and target rectangles shall remain unchanged across closing and reopening a device, such that piping data into or out of a device will work without special preparations. More advanced applications should ensure the parameters are suitable before starting I/O.

Note:

On the next two examples, a video capture device is assumed; change V4L2_BUF_TYPE_VIDEO_CAPTURE for other types of device.

Example: Resetting the cropping parameters

```
struct v4l2_cropcap cropcap;
struct v4l2_crop crop;

memset (&cropcap, 0, sizeof (cropcap));
cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_CROPCAP, &cropcap)) {
    perror ("VIDIOC_CROPCAP");
    exit (EXIT_FAILURE);
}

memset (&crop, 0, sizeof (crop));
crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
crop.c = cropcap.defrect;

/* Ignore if cropping is not supported (EINVAL). */

if (-1 == ioctl (fd, VIDIOC_S_CROP, &crop)
    && errno != EINVAL) {
    perror ("VIDIOC_S_CROP");
    exit (EXIT_FAILURE);
}
```

Example: Simple downscaling

```
struct v4l2_cropcap cropcap;
struct v4l2_format format;
```

```
reset_cropping_parameters ();

/* Scale down to 1/4 size of full picture. */

memset (&format, 0, sizeof (format)); /* defaults */

format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

format.fmt.pix.width = cropcap.defrect.width >> 1;
format.fmt.pix.height = cropcap.defrect.height >> 1;
format.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;

if (-1 == ioctl (fd, VIDIOC_S_FMT, &format)) {
    perror ("VIDIOC_S_FORMAT");
    exit (EXIT_FAILURE);
}

/* We could check the actual image size now, the actual scaling factor
   or if the driver can scale at all. */
```

Example: Selecting an output area

Note:

This example assumes an output device.

```
struct v4l2_cropcap cropcap;
struct v4l2_crop crop;

memset (&cropcap, 0, sizeof (cropcap));
cropcap.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;

if (-1 == ioctl (fd, VIDIOC_CROPCAP;, &cropcap)) {
    perror ("VIDIOC_CROPCAP");
    exit (EXIT_FAILURE);
}

memset (&crop, 0, sizeof (crop));

crop.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
crop.c = cropcap.defrect;

/* Scale the width and height to 50 % of their original size
   and center the output. */

crop.c.width /= 2;
crop.c.height /= 2;
crop.c.left += crop.c.width / 2;
crop.c.top += crop.c.height / 2;

/* Ignore if cropping is not supported (EINVAL). */

if (-1 == ioctl (fd, VIDIOC_S_CROP, &crop)
    && errno != EINVAL) {
    perror ("VIDIOC_S_CROP");
    exit (EXIT_FAILURE);
}
```

Example: Current scaling factor and pixel aspect

Note:

This example assumes a video capture device.

```
struct v4l2_cropcap cropcap;
struct v4l2_crop crop;
struct v4l2_format format;
double hscale, vscale;
double aspect;
int dwidth, dheight;

memset (&cropcap, 0, sizeof (cropcap));
cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_CROPCAP, &cropcap)) {
    perror ("VIDIOC_CROPCAP");
    exit (EXIT_FAILURE);
}

memset (&crop, 0, sizeof (crop));
crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_G_CROP, &crop)) {
    if (errno != EINVAL) {
        perror ("VIDIOC_G_CROP");
        exit (EXIT_FAILURE);
    }

    /* Cropping not supported. */
    crop.c = cropcap.defrect;
}

memset (&format, 0, sizeof (format));
format.fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_G_FMT, &format)) {
    perror ("VIDIOC_G_FMT");
    exit (EXIT_FAILURE);
}

/* The scaling applied by the driver. */

hscale = format.fmt.pix.width / (double) crop.c.width;
vscale = format.fmt.pix.height / (double) crop.c.height;

aspect = cropcap.pixelaspect.numerator /
     (double) cropcap.pixelaspect.denominator;
aspect = aspect * hscale / vscale;

/* Devices following ITU-R BT.601 do not capture
   square pixels. For playback on a computer monitor
   we should scale the images to this size. */

dwidth = format.fmt.pix.width / aspect;
dheight = format.fmt.pix.height;
```

API for cropping, composing and scaling

Introduction

Some video capture devices can sample a subsection of a picture and shrink or enlarge it to an image of arbitrary size. Next, the devices can insert the image into larger one. Some video output devices can crop part of an input image, scale it up or down and insert it at an arbitrary scan line and horizontal offset into a video signal. We call these abilities cropping, scaling and composing.

On a video *capture* device the source is a video signal, and the cropping target determine the area actually sampled. The sink is an image stored in a memory buffer. The composing area specifies which part of the buffer is actually written to by the hardware.

On a video *output* device the source is an image in a memory buffer, and the cropping target is a part of an image to be shown on a display. The sink is the display or the graphics screen. The application may select the part of display where the image should be displayed. The size and position of such a window is controlled by the compose target.

Rectangles for all cropping and composing targets are defined even if the device does supports neither cropping nor composing. Their size and position will be fixed in such a case. If the device does not support scaling then the cropping and composing rectangles have the same size.

Selection targets

Fig. 1.3: Cropping and composing targets
Targets used by a cropping, composing and scaling process

See *Selection targets* for more information.

Configuration

Applications can use the *selection API* to select an area in a video signal or a buffer, and to query for default settings and hardware limits.

Video hardware can have various cropping, composing and scaling limitations. It may only scale up or down, support only discrete scaling factors, or have different scaling abilities in the horizontal and vertical directions. Also it may not support scaling at all. At the same time the cropping/composing rectangles may have to be aligned, and both the source and the sink may have arbitrary upper and lower size limits. Therefore, as usual, drivers are expected to adjust the requested parameters and return the actual values selected. An application can control the rounding behaviour using *constraint flags* .

Configuration of video capture

See figure *Cropping and composing targets* for examples of the selection targets available for a video capture device. It is recommended to configure the cropping targets before to the composing targets.

The range of coordinates of the top left corner, width and height of areas that can be sampled is given by the `V4L2_SEL_TGT_CROP_BOUNDS` target. It is recommended for the driver developers to put the top/left corner at position `(0,0)`. The rectangle's coordinates are expressed in pixels.

The top left corner, width and height of the source rectangle, that is the area actually sampled, is given by the `V4L2_SEL_TGT_CROP` target. It uses the same coordinate system as `V4L2_SEL_TGT_CROP_BOUNDS`. The active cropping area must lie completely inside the capture boundaries. The driver may further adjust the requested size and/or position according to hardware limitations.

Each capture device has a default source rectangle, given by the V4L2_SEL_TGT_CROP_DEFAULT target. This rectangle shall over what the driver writer considers the complete picture. Drivers shall set the active crop rectangle to the default when the driver is first loaded, but not later.

The composing targets refer to a memory buffer. The limits of composing coordinates are obtained using V4L2_SEL_TGT_COMPOSE_BOUNDS. All coordinates are expressed in pixels. The rectangle's top/left corner must be located at position (0,0). The width and height are equal to the image size set by *VIDIOC_S_FMT* .

The part of a buffer into which the image is inserted by the hardware is controlled by the V4L2_SEL_TGT_COMPOSE target. The rectangle's coordinates are also expressed in the same coordinate system as the bounds rectangle. The composing rectangle must lie completely inside bounds rectangle. The driver must adjust the composing rectangle to fit to the bounding limits. Moreover, the driver can perform other adjustments according to hardware limitations. The application can control rounding behaviour using *constraint flags* .

For capture devices the default composing rectangle is queried using V4L2_SEL_TGT_COMPOSE_DEFAULT. It is usually equal to the bounding rectangle.

The part of a buffer that is modified by the hardware is given by V4L2_SEL_TGT_COMPOSE_PADDED. It contains all pixels defined using V4L2_SEL_TGT_COMPOSE plus all padding data modified by hardware during insertion process. All pixels outside this rectangle *must not* be changed by the hardware. The content of pixels that lie inside the padded area but outside active area is undefined. The application can use the padded and active rectangles to detect where the rubbish pixels are located and remove them if needed.

Configuration of video output

For output devices targets and ioctls are used similarly to the video capture case. The *composing* rectangle refers to the insertion of an image into a video signal. The cropping rectangles refer to a memory buffer. It is recommended to configure the composing targets before to the cropping targets.

The cropping targets refer to the memory buffer that contains an image to be inserted into a video signal or graphical screen. The limits of cropping coordinates are obtained using V4L2_SEL_TGT_CROP_BOUNDS. All coordinates are expressed in pixels. The top/left corner is always point (0,0). The width and height is equal to the image size specified using *VIDIOC_S_FMT* ioctl.

The top left corner, width and height of the source rectangle, that is the area from which image date are processed by the hardware, is given by the V4L2_SEL_TGT_CROP. Its coordinates are expressed in in the same coordinate system as the bounds rectangle. The active cropping area must lie completely inside the crop boundaries and the driver may further adjust the requested size and/or position according to hardware limitations.

For output devices the default cropping rectangle is queried using V4L2_SEL_TGT_CROP_DEFAULT. It is usually equal to the bounding rectangle.

The part of a video signal or graphics display where the image is inserted by the hardware is controlled by V4L2_SEL_TGT_COMPOSE target. The rectangle's coordinates are expressed in pixels. The composing rectangle must lie completely inside the bounds rectangle. The driver must adjust the area to fit to the bounding limits. Moreover, the driver can perform other adjustments according to hardware limitations.

The device has a default composing rectangle, given by the V4L2_SEL_TGT_COMPOSE_DEFAULT target. This rectangle shall cover what the driver writer considers the complete picture. It is recommended for the driver developers to put the top/left corner at position (0,0). Drivers shall set the active composing rectangle to the default one when the driver is first loaded.

The devices may introduce additional content to video signal other than an image from memory buffers. It includes borders around an image. However, such a padded area is driver-dependent feature not covered by this document. Driver developers are encouraged to keep padded rectangle equal to active one. The padded target is accessed by the V4L2_SEL_TGT_COMPOSE_PADDED identifier. It must contain all pixels from the V4L2_SEL_TGT_COMPOSE target.

Scaling control

An application can detect if scaling is performed by comparing the width and the height of rectangles obtained using V4L2_SEL_TGT_CROP and V4L2_SEL_TGT_COMPOSE targets. If these are not equal then the scaling is applied. The application can compute the scaling ratios using these values.

Comparison with old cropping API

The selection API was introduced to cope with deficiencies of previous *API* , that was designed to control simple capture devices. Later the cropping API was adopted by video output drivers. The ioctls are used to select a part of the display were the video signal is inserted. It should be considered as an API abuse because the described operation is actually the composing. The selection API makes a clear distinction between composing and cropping operations by setting the appropriate targets. The V4L2 API lacks any support for composing to and cropping from an image inside a memory buffer. The application could configure a capture device to fill only a part of an image by abusing V4L2 API. Cropping a smaller image from a larger one is achieved by setting the field `bytesperline` at struct *v4l2_pix_format*. Introducing an image offsets could be done by modifying field `m_userptr` at struct *v4l2_buffer* before calling *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* . Those operations should be avoided because they are not portable (endianness), and do not work for macroblock and Bayer formats and mmap buffers. The selection API deals with configuration of buffer cropping/composing in a clear, intuitive and portable way. Next, with the selection API the concepts of the padded target and constraints flags are introduced. Finally, struct *v4l2_crop* and struct *v4l2_cropcap* have no reserved fields. Therefore there is no way to extend their functionality. The new struct *v4l2_selection* provides a lot of place for future extensions. Driver developers are encouraged to implement only selection API. The former cropping API would be simulated using the new one.

Examples

(A video capture device is assumed; change V4L2_BUF_TYPE_VIDEO_CAPTURE for other devices; change target to V4L2_SEL_TGT_COMPOSE_* family to configure composing area)

Example: Resetting the cropping parameters

```
struct v4l2_selection sel = {
    .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,
    .target = V4L2_SEL_TGT_CROP_DEFAULT,
};
ret = ioctl(fd, VIDIOC_G_SELECTION, &sel);
if (ret)
    exit(-1);
sel.target = V4L2_SEL_TGT_CROP;
ret = ioctl(fd, VIDIOC_S_SELECTION, &sel);
if (ret)
    exit(-1);
```

Setting a composing area on output of size of *at most* half of limit placed at a center of a display.

Example: Simple downscaling

```
struct v4l2_selection sel = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .target = V4L2_SEL_TGT_COMPOSE_BOUNDS,
};
struct v4l2_rect r;
```

```
ret = ioctl(fd, VIDIOC_G_SELECTION, &sel);
if (ret)
    exit(-1);
/* setting smaller compose rectangle */
r.width = sel.r.width / 2;
r.height = sel.r.height / 2;
r.left = sel.r.width / 4;
r.top = sel.r.height / 4;
sel.r = r;
sel.target = V4L2_SEL_TGT_COMPOSE;
sel.flags = V4L2_SEL_FLAG_LE;
ret = ioctl(fd, VIDIOC_S_SELECTION, &sel);
if (ret)
    exit(-1);
```

A video output device is assumed; change V4L2_BUF_TYPE_VIDEO_OUTPUT for other devices

Example: Querying for scaling factors

```
struct v4l2_selection compose = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .target = V4L2_SEL_TGT_COMPOSE,
};
struct v4l2_selection crop = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .target = V4L2_SEL_TGT_CROP,
};
double hscale, vscale;

ret = ioctl(fd, VIDIOC_G_SELECTION, &compose);
if (ret)
    exit(-1);
ret = ioctl(fd, VIDIOC_G_SELECTION, &crop);
if (ret)
    exit(-1);

/* computing scaling factors */
hscale = (double)compose.r.width / crop.r.width;
vscale = (double)compose.r.height / crop.r.height;
```

Streaming Parameters

Streaming parameters are intended to optimize the video capture process as well as I/O. Presently applications can request a high quality capture mode with the *VIDIOC_S_PARM* ioctl.

The current video standard determines a nominal number of frames per second. If less than this number of frames is to be captured or output, applications can request frame skipping or duplicating on the driver side. This is especially useful when using the *read()* or *write()* , which are not augmented by timestamps or sequence counters, and to avoid unnecessary data copying.

Finally these ioctls can be used to determine the number of buffers used internally by a driver in read/write mode. For implications see the section discussing the *read()* function.

To get and set the streaming parameters applications call the *VIDIOC_G_PARM* and *VIDIOC_S_PARM* ioctl, respectively. They take a pointer to a struct *v4l2_streamparm*, which contains a union holding separate parameters for input and output devices.

These ioctls are optional, drivers need not implement them. If so, they return the EINVAL error code.

1.2.2 Image Formats

The V4L2 API was primarily designed for devices exchanging image data with applications. The struct *v4l2_pix_format* and struct *v4l2_pix_format_mplane* structures define the format and layout of an image in memory. The former is used with the single-planar API, while the latter is used with the multi-planar version (see *Single- and multi-planar APIs*). Image formats are negotiated with the *VIDIOC_S_FMT* ioctl. (The explanations here focus on video capturing and output, for overlay frame buffer formats see also *VIDIOC_G_FBUF* .)

Single-planar format structure

v4l2_pix_format

Table 1.1: struct v4l2_pix_format

__u32	width	Image width in pixels.
__u32	height	Image height in pixels. If field is one of V4L2_FIELD_TOP, V4L2_FIELD_BOTTOM or V4L2_FIELD_ALTERNATE then height refers to the number of lines in the field, otherwise it refers to the number of lines in the frame (which is twice the field height for interlaced formats).
Applications set these fields to request an image size, drivers return the closest possible values. In case of planar formats the width and height applies to the largest plane. To avoid ambiguities drivers must return values rounded up to a multiple of the scale factor of any smaller planes. For example when the image format is YUV 4:2:0, width and height must be multiples of two.		
__u32	pixelformat	The pixel format or type of compression, set by the application. This is a little endian *four character code* . V4L2 defines standard RGB formats in *Packed RGB Image Formats* , YUV formats in *YUV Formats* , and reserved codes in *Reserved Image Formats*
enum :c:type::*v4l2_field*	field	Video images are typically interlaced. Applications can request to capture or output only the top or bottom field, or both fields interlaced or sequentially stored in one buffer or alternating in separate buffers. Drivers return the actual field order selected. For more details on fields see *Field Order* .
__u32	bytesperline	Distance in bytes between the leftmost pixels in two adjacent lines.

Continued on next page

Table 1.1 – continued from previous page

		Both applications and drivers can set this field to request padding bytes at the end of each line. Drivers however may ignore the value requested by the application, returning `width` times bytes per pixel or a larger value required by the hardware. That implies applications can just set this field to zero to get a reasonable default. Video hardware may access padding bytes, therefore they must reside in accessible memory. Consider cases where padding bytes after the last line of an image cross a system page boundary. Input devices may write padding bytes, the value is undefined. Output devices ignore the contents of padding bytes. When the image format is planar the `bytesperline` value applies to the first plane and is divided by the same factor as the `width` field for the other planes. For example the Cb and Cr planes of a YUV 4:2:0 image have half as many padding bytes following each line as the Y plane. To avoid ambiguities drivers must return a `bytesperline` value rounded up to a multiple of the scale factor. For compressed formats the `bytesperline` value makes no sense. Applications and drivers must set this to 0 in that case.
__u32	`sizeimage`	Size in bytes of the buffer to hold a complete image, set by the driver. Usually this is `bytesperline` times `height`. When the image consists of variable length compressed data this is the maximum number of bytes required to hold an image.
enum *v4l2_colorspace*	`colorspace`	This information supplements the `pixelformat` and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
__u32	`priv`	This field indicates whether the remaining fields of the struct *v4l2_pix_format*, also called the extended fields, are valid. When set to `V4L2_PIX_FMT_PRIV_MAGIC`, it indicates that the extended fields have been correctly initialized. When set to any other value it indicates that the extended fields contain undefined values. Applications that wish to use the pixel format extended fields must first ensure that the feature is supported by querying the device for the *V4L2_CAP_EXT_PIX_FORMAT* capability. If the capability isn't set the pixel format extended fields are not supported and using the extended fields will lead to undefined results. To use the extended fields, applications must set the `priv` field to `V4L2_PIX_FMT_PRIV_MAGIC`, initialize all the extended fields and zero the unused bytes of the struct *v4l2_format* `raw_data` field. When the `priv` field isn't set to `V4L2_PIX_FMT_PRIV_MAGIC` drivers must act as if all the extended fields were set to zero. On return drivers must set the `priv` field to `V4L2_PIX_FMT_PRIV_MAGIC` and all the extended fields to applicable values.
__u32	`flags`	Flags set by the application or driver, see *Format Flags* .

Continued on next page

Table 1.1 – continued from previous page

enum *v4l2_ycbcr_encoding*	ycbcr_enc	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
enum *v4l2_hsv_encoding*	hsv_enc	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
enum *v4l2_quantization*	quantization	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
enum *v4l2_xfer_func*	xfer_func	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .

Multi-planar format structures

The struct *v4l2_plane_pix_format* structures define size and layout for each of the planes in a multi-planar format. The struct *v4l2_pix_format_mplane* structure contains information common to all planes (such as image width and height) and an array of struct *v4l2_plane_pix_format* structures, describing all planes of that format.

v4l2_plane_pix_format

Table 1.2: struct v4l2_plane_pix_format

__u32	sizeimage	Maximum size in bytes required for image data in this plane.
__u32	bytesperline	Distance in bytes between the leftmost pixels in two adjacent lines. See struct *v4l2_pix_format*.
__u16	reserved[6]	Reserved for future extensions. Should be zeroed by drivers and applications.

v4l2_pix_format_mplane

Table 1.3: struct v4l2_pix_format_mplane

__u32	width	Image width in pixels. See struct *v4l2_pix_format*.
__u32	height	Image height in pixels. See struct *v4l2_pix_format*.
__u32	pixelformat	The pixel format. Both single- and multi-planar four character codes can be used.
enum *v4l2_field*	field	See struct *v4l2_pix_format*.
enum *v4l2_colorspace*	colorspace	See struct *v4l2_pix_format*.
struct *v4l2_plane_pix_format*	plane_fmt[VIDEO_MAX_PLANES]	An array of structures describing format of each plane this pixel format consists of. The number of valid entries in this array has to be put in the num_planes field.
__u8	num_planes	Number of planes (i.e. separate memory buffers) for this format and the number of valid entries in the plane_fmt array.
__u8	flags	Flags set by the application or driver, see *Format Flags* .
enum *v4l2_ycbcr_encoding*	ycbcr_enc	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
enum *v4l2_hsv_encoding*	hsv_enc	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
enum *v4l2_quantization*	quantization	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
enum *v4l2_xfer_func*	xfer_func	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
__u8	reserved[7]	Reserved for future extensions. Should be zeroed by drivers and applications.

Standard Image Formats

In order to exchange images between drivers and applications, it is necessary to have standard image data formats which both sides will interpret the same way. V4L2 includes several such formats, and this section is intended to be an unambiguous specification of the standard image data formats in V4L2.

V4L2 drivers are not limited to these formats, however. Driver-specific formats are possible. In that case the application may depend on a codec to convert images to one of the standard formats when needed. But the data can still be stored and retrieved in the proprietary format. For example, a device may support a proprietary compressed format. Applications can still capture and save the data in the compressed format, saving much disk space, and later use a codec to convert the images to the X Windows screen format when the video is to be displayed.

Even so, ultimately, some standard formats are needed, so the V4L2 specification would not be complete without well-defined standard formats.

The V4L2 standard formats are mainly uncompressed formats. The pixels are always arranged in memory from left to right, and from top to bottom. The first byte of data in the image buffer is always for the leftmost pixel of the topmost row. Following that is the pixel immediately to its right, and so on until the end of the top row of pixels. Following the rightmost pixel of the row there may be zero or more bytes of

padding to guarantee that each row of pixel data has a certain alignment. Following the pad bytes, if any, is data for the leftmost pixel of the second row from the top, and so on. The last row has just as many pad bytes after it as the other rows.

In V4L2 each format has an identifier which looks like PIX_FMT_XXX, defined in the *videodev2.h* header file. These identifiers represent *four character (FourCC) codes* which are also listed below, however they are not the same as those used in the Windows world.

For some formats, data is stored in separate, discontiguous memory buffers. Those formats are identified by a separate set of FourCC codes and are referred to as "multi-planar formats". For example, a *YUV422* frame is normally stored in one memory buffer, but it can also be placed in two or three separate buffers, with Y component in one buffer and CbCr components in another in the 2-planar version or with each component in its own buffer in the 3-planar case. Those sub-buffers are referred to as "*planes*".

Colorspaces

'Color' is a very complex concept and depends on physics, chemistry and biology. Just because you have three numbers that describe the 'red', 'green' and 'blue' components of the color of a pixel does not mean that you can accurately display that color. A colorspace defines what it actually *means* to have an RGB value of e.g. (255, 0, 0). That is, which color should be reproduced on the screen in a perfectly calibrated environment.

In order to do that we first need to have a good definition of color, i.e. some way to uniquely and unambiguously define a color so that someone else can reproduce it. Human color vision is trichromatic since the human eye has color receptors that are sensitive to three different wavelengths of light. Hence the need to use three numbers to describe color. Be glad you are not a mantis shrimp as those are sensitive to 12 different wavelengths, so instead of RGB we would be using the ABCDEFGHIJKL colorspace...

Color exists only in the eye and brain and is the result of how strongly color receptors are stimulated. This is based on the Spectral Power Distribution (SPD) which is a graph showing the intensity (radiant power) of the light at wavelengths covering the visible spectrum as it enters the eye. The science of colorimetry is about the relationship between the SPD and color as perceived by the human brain.

Since the human eye has only three color receptors it is perfectly possible that different SPDs will result in the same stimulation of those receptors and are perceived as the same color, even though the SPD of the light is different.

In the 1920s experiments were devised to determine the relationship between SPDs and the perceived color and that resulted in the CIE 1931 standard that defines spectral weighting functions that model the perception of color. Specifically that standard defines functions that can take an SPD and calculate the stimulus for each color receptor. After some further mathematical transforms these stimuli are known as the *CIE XYZ tristimulus* values and these X, Y and Z values describe a color as perceived by a human unambiguously. These X, Y and Z values are all in the range [0…1].

The Y value in the CIE XYZ colorspace corresponds to luminance. Often the CIE XYZ colorspace is transformed to the normalized CIE xyY colorspace:

x = X / (X + Y + Z)

y = Y / (X + Y + Z)

The x and y values are the chromaticity coordinates and can be used to define a color without the luminance component Y. It is very confusing to have such similar names for these colorspaces. Just be aware that if colors are specified with lower case 'x' and 'y', then the CIE xyY colorspace is used. Upper case 'X' and 'Y' refer to the CIE XYZ colorspace. Also, y has nothing to do with luminance. Together x and y specify a color, and Y the luminance. That is really all you need to remember from a practical point of view. At the end of this section you will find reading resources that go into much more detail if you are interested.

A monitor or TV will reproduce colors by emitting light at three different wavelengths, the combination of which will stimulate the color receptors in the eye and thus cause the perception of color. Historically these wavelengths were defined by the red, green and blue phosphors used in the displays. These *color primaries* are part of what defines a colorspace.

Different display devices will have different primaries and some primaries are more suitable for some display technologies than others. This has resulted in a variety of colorspaces that are used for different display technologies or uses. To define a colorspace you need to define the three color primaries (these are typically defined as x, y chromaticity coordinates from the CIE xyY colorspace) but also the white reference: that is the color obtained when all three primaries are at maximum power. This determines the relative power or energy of the primaries. This is usually chosen to be close to daylight which has been defined as the CIE D65 Illuminant.

To recapitulate: the CIE XYZ colorspace uniquely identifies colors. Other colorspaces are defined by three chromaticity coordinates defined in the CIE xyY colorspace. Based on those a 3x3 matrix can be constructed that transforms CIE XYZ colors to colors in the new colorspace.

Both the CIE XYZ and the RGB colorspace that are derived from the specific chromaticity primaries are linear colorspaces. But neither the eye, nor display technology is linear. Doubling the values of all components in the linear colorspace will not be perceived as twice the intensity of the color. So each colorspace also defines a transfer function that takes a linear color component value and transforms it to the non-linear component value, which is a closer match to the non-linear performance of both the eye and displays. Linear component values are denoted RGB, non-linear are denoted as R'G'B'. In general colors used in graphics are all R'G'B', except in openGL which uses linear RGB. Special care should be taken when dealing with openGL to provide linear RGB colors or to use the built-in openGL support to apply the inverse transfer function.

The final piece that defines a colorspace is a function that transforms non-linear R'G'B' to non-linear Y'CbCr. This function is determined by the so-called luma coefficients. There may be multiple possible Y'CbCr encodings allowed for the same colorspace. Many encodings of color prefer to use luma (Y') and chroma (CbCr) instead of R'G'B'. Since the human eye is more sensitive to differences in luminance than in color this encoding allows one to reduce the amount of color information compared to the luma data. Note that the luma (Y') is unrelated to the Y in the CIE XYZ colorspace. Also note that Y'CbCr is often called YCbCr or YUV even though these are strictly speaking wrong.

Sometimes people confuse Y'CbCr as being a colorspace. This is not correct, it is just an encoding of an R'G'B' color into luma and chroma values. The underlying colorspace that is associated with the R'G'B' color is also associated with the Y'CbCr color.

The final step is how the RGB, R'G'B' or Y'CbCr values are quantized. The CIE XYZ colorspace where X, Y and Z are in the range [0…1] describes all colors that humans can perceive, but the transform to another colorspace will produce colors that are outside the [0…1] range. Once clamped to the [0…1] range those colors can no longer be reproduced in that colorspace. This clamping is what reduces the extent or gamut of the colorspace. How the range of [0…1] is translated to integer values in the range of [0…255] (or higher, depending on the color depth) is called the quantization. This is *not* part of the colorspace definition. In practice RGB or R'G'B' values are full range, i.e. they use the full [0…255] range. Y'CbCr values on the other hand are limited range with Y' using [16…235] and Cb and Cr using [16…240].

Unfortunately, in some cases limited range RGB is also used where the components use the range [16…235]. And full range Y'CbCr also exists using the [0…255] range.

In order to correctly interpret a color you need to know the quantization range, whether it is R'G'B' or Y'CbCr, the used Y'CbCr encoding and the colorspace. From that information you can calculate the corresponding CIE XYZ color and map that again to whatever colorspace your display device uses.

The colorspace definition itself consists of the three chromaticity primaries, the white reference chromaticity, a transfer function and the luma coefficients needed to transform R'G'B' to Y'CbCr. While some colorspace standards correctly define all four, quite often the colorspace standard only defines some, and you have to rely on other standards for the missing pieces. The fact that colorspaces are often a mix of different standards also led to very confusing naming conventions where the name of a standard was used to name a colorspace when in fact that standard was part of various other colorspaces as well.

If you want to read more about colors and colorspaces, then the following resources are useful: *poynton* is a good practical book for video engineers, *colimg* has a much broader scope and describes many more aspects of color (physics, chemistry, biology, etc.). The http://www.brucelindbloom.com website is an excellent resource, especially with respect to the mathematics behind colorspace conversions. The wikipedia CIE 1931 colorspace article is also very useful.

Defining Colorspaces in V4L2

In V4L2 colorspaces are defined by four values. The first is the colorspace identifier (enum *v4l2_colorspace*) which defines the chromaticities, the default transfer function, the default Y'CbCr encoding and the default quantization method. The second is the transfer function identifier (enum *v4l2_xfer_func*) to specify non-standard transfer functions. The third is the Y'CbCr encoding identifier (enum *v4l2_ycbcr_encoding*) to specify non-standard Y'CbCr encodings and the fourth is the quantization identifier (enum *v4l2_quantization*) to specify non-standard quantization methods. Most of the time only the colorspace field of struct *v4l2_pix_format* or struct *v4l2_pix_format_mplane* needs to be filled in. On *HSV formats* the *Hue* is defined as the angle on the cylindrical color representation. Usually this angle is measured in degrees, i.e. 0-360. When we map this angle value into 8 bits, there are two basic ways to do it: Divide the angular value by 2 (0-179), or use the whole range, 0-255, dividing the angular value by 1.41. The enum *v4l2_hsv_encoding* specifies which encoding is used.

Note:

The default R'G'B' quantization is full range for all colorspaces except for BT.2020 which uses limited range R'G'B' quantization.

v4l2_colorspace

Table 1.4: V4L2 Colorspaces

Identifier	Details
V4L2_COLORSPACE_DEFAULT	The default colorspace. This can be used by applications to let the driver fill in the colorspace.
V4L2_COLORSPACE_SMPTE170M	See *Colorspace SMPTE 170M (V4L2_COLORSPACE_SMPTE170M)* .
V4L2_COLORSPACE_REC709	See *Colorspace Rec. 709 (V4L2_COLORSPACE_REC709)* .
V4L2_COLORSPACE_SRGB	See *Colorspace sRGB (V4L2_COLORSPACE_SRGB)* .
V4L2_COLORSPACE_ADOBERGB	See *Colorspace Adobe RGB (V4L2_COLORSPACE_ADOBERGB)* .
V4L2_COLORSPACE_BT2020	See *Colorspace BT.2020 (V4L2_COLORSPACE_BT2020)* .
V4L2_COLORSPACE_DCI_P3	See *Colorspace DCI-P3 (V4L2_COLORSPACE_DCI_P3)* .
V4L2_COLORSPACE_SMPTE240M	See *Colorspace SMPTE 240M (V4L2_COLORSPACE_SMPTE240M)* .
V4L2_COLORSPACE_470_SYSTEM_M	See *Colorspace NTSC 1953 (V4L2_COLORSPACE_470_SYSTEM_M)* .
V4L2_COLORSPACE_470_SYSTEM_BG	See *Colorspace EBU Tech. 3213 (V4L2_COLORSPACE_470_SYSTEM_BG)* .
V4L2_COLORSPACE_JPEG	See *Colorspace JPEG (V4L2_COLORSPACE_JPEG)* .
V4L2_COLORSPACE_RAW	The raw colorspace. This is used for raw image capture where the image is minimally processed and is using the internal colorspace of the device. The software that processes an image using this 'colorspace' will have to know the internals of the capture device.

v4l2_xfer_func

Table 1.5: V4L2 Transfer Function

Identifier	Details
V4L2_XFER_FUNC_DEFAULT	Use the default transfer function as defined by the colorspace.
V4L2_XFER_FUNC_709	Use the Rec. 709 transfer function.
V4L2_XFER_FUNC_SRGB	Use the sRGB transfer function.
V4L2_XFER_FUNC_ADOBERGB	Use the AdobeRGB transfer function.
V4L2_XFER_FUNC_SMPTE240M	Use the SMPTE 240M transfer function.
V4L2_XFER_FUNC_NONE	Do not use a transfer function (i.e. use linear RGB values).
V4L2_XFER_FUNC_DCI_P3	Use the DCI-P3 transfer function.
V4L2_XFER_FUNC_SMPTE2084	Use the SMPTE 2084 transfer function.

v4l2_ycbcr_encoding

Table 1.6: V4L2 Y'CbCr Encodings

Identifier	Details
V4L2_YCBCR_ENC_DEFAULT	Use the default Y'CbCr encoding as defined by the colorspace.
V4L2_YCBCR_ENC_601	Use the BT.601 Y'CbCr encoding.
V4L2_YCBCR_ENC_709	Use the Rec. 709 Y'CbCr encoding.
V4L2_YCBCR_ENC_XV601	Use the extended gamut xvYCC BT.601 encoding.
V4L2_YCBCR_ENC_XV709	Use the extended gamut xvYCC Rec. 709 encoding.
V4L2_YCBCR_ENC_BT2020	Use the default non-constant luminance BT.2020 Y'CbCr encoding.
V4L2_YCBCR_ENC_BT2020_CONST_LUM	Use the constant luminance BT.2020 Yc'CbcCrc encoding.
V4L2_YCBCR_ENC_SMPTE_240M	Use the SMPTE 240M Y'CbCr encoding.

v4l2_hsv_encoding

Table 1.7: V4L2 HSV Encodings

Identifier	Details
V4L2_HSV_ENC_180	For the Hue, each LSB is two degrees.
V4L2_HSV_ENC_256	For the Hue, the 360 degrees are mapped into 8 bits, i.e. each LSB is roughly 1.41 degrees.

v4l2_quantization

Table 1.8: V4L2 Quantization Methods

Identifier	Details
V4L2_QUANTIZATION_DEFAULT	Use the default quantization encoding as defined by the colorspace. This is always full range for R'G'B' (except for the BT.2020 colorspace) and HSV. It is usually limited range for Y'CbCr.
V4L2_QUANTIZATION_FULL_RANGE	Use the full range quantization encoding. I.e. the range [0…1] is mapped to [0…255] (with possible clipping to [1…254] to avoid the 0x00 and 0xff values). Cb and Cr are mapped from [-0.5…0.5] to [0…255] (with possible clipping to [1…254] to avoid the 0x00 and 0xff values).
V4L2_QUANTIZATION_LIM_RANGE	Use the limited range quantization encoding. I.e. the range [0…1] is mapped to [16…235]. Cb and Cr are mapped from [-0.5…0.5] to [16…240].

Detailed Colorspace Descriptions

Colorspace SMPTE 170M (V4L2_COLORSPACE_SMPTE170M)

The *SMPTE 170M* standard defines the colorspace used by NTSC and PAL and by SDTV in general. The default transfer function is V4L2_XFER_FUNC_709. The default Y'CbCr encoding is V4L2_YCBCR_ENC_601. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 1.9: SMPTE 170M Chromaticities

Color	x	y
Red	0.630	0.340
Green	0.310	0.595
Blue	0.155	0.070
White Reference (D65)	0.3127	0.3290

The red, green and blue chromaticities are also often referred to as the SMPTE C set, so this colorspace is sometimes called SMPTE C as well.

The transfer function defined for SMPTE 170M is the same as the one defined in Rec. 709.

$$L' = -1.099(-L)^{0.45} + 0.099, \text{ for } L \leq -0.018$$
$$L' = 4.5L, \text{ for } -0.018 < L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099, \text{ for } L \geq 0.018$$

Inverse Transfer function:

$$L = -\left(\frac{L' - 0.099}{-1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \leq -0.081$$
$$L = \frac{L'}{4.5}, \text{ for } -0.081 < L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. This conversion to Y'CbCr is identical to the one defined in the *ITU BT.601* standard and this colorspace is sometimes called BT.601 as well, even though BT.601 does not mention any color primaries.

The default quantization is limited range, but full range is possible although rarely seen.

Colorspace Rec. 709 (V4L2_COLORSPACE_REC709)

The *ITU BT.709* standard defines the colorspace used by HDTV in general. The default transfer function is V4L2_XFER_FUNC_709. The default Y'CbCr encoding is V4L2_YCBCR_ENC_709. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 1.10: Rec. 709 Chromaticities

Color	x	y
Red	0.640	0.330
Green	0.300	0.600
Blue	0.150	0.060
White Reference (D65)	0.3127	0.3290

The full name of this standard is Rec. ITU-R BT.709-5.

Transfer function. Normally L is in the range [0…1], but for the extended gamut xvYCC encoding values

outside that range are allowed.

$$L' = -1.099(-L)^{0.45} + 0.099, \text{ for } L \leq -0.018$$
$$L' = 4.5L, \text{ for } -0.018 < L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099, \text{ for } L \geq 0.018$$

Inverse Transfer function:

$$L = -\left(\frac{L' - 0.099}{-1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \leq -0.081$$
$$L = \frac{L'}{4.5}, \text{ for } -0.081 < L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_709 encoding:

$$Y' = 0.2126R' + 0.7152G' + 0.0722B'$$
$$Cb = -0.1146R' - 0.3854G' + 0.5B'$$
$$Cr = 0.5R' - 0.4542G' - 0.0458B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5].

The default quantization is limited range, but full range is possible although rarely seen.

The V4L2_YCBCR_ENC_709 encoding described above is the default for this colorspace, but it can be overridden with V4L2_YCBCR_ENC_601, in which case the BT.601 Y'CbCr encoding is used.

Two additional extended gamut Y'CbCr encodings are also possible with this colorspace:

The xvYCC 709 encoding (V4L2_YCBCR_ENC_XV709, *xvYCC*) is similar to the Rec. 709 encoding, but it allows for R', G' and B' values that are outside the range [0…1]. The resulting Y', Cb and Cr values are scaled and offset:

$$Y' = \frac{219}{256} * (0.2126R' + 0.7152G' + 0.0722B') + \frac{16}{256}$$
$$Cb = \frac{224}{256} * (-0.1146R' - 0.3854G' + 0.5B')$$
$$Cr = \frac{224}{256} * (0.5R' - 0.4542G' - 0.0458B')$$

The xvYCC 601 encoding (V4L2_YCBCR_ENC_XV601, *xvYCC*) is similar to the BT.601 encoding, but it allows for R', G' and B' values that are outside the range [0…1]. The resulting Y', Cb and Cr values are scaled and offset:

$$Y' = \frac{219}{256} * (0.2990R' + 0.5870G' + 0.1140B') + \frac{16}{256}$$
$$Cb = \frac{224}{256} * (-0.1687R' - 0.3313G' + 0.5B')$$
$$Cr = \frac{224}{256} * (0.5R' - 0.4187G' - 0.0813B')$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. The non-standard xvYCC 709 or xvYCC 601 encodings can be used by selecting V4L2_YCBCR_ENC_XV709 or V4L2_YCBCR_ENC_XV601. The xvYCC encodings always use full range quantization.

Colorspace sRGB (V4L2_COLORSPACE_SRGB)

The *sRGB* standard defines the colorspace used by most webcams and computer graphics. The default transfer function is V4L2_XFER_FUNC_SRGB. The default Y'CbCr encoding is V4L2_YCBCR_ENC_601. The

default Y'CbCr quantization is full range. The chromaticities of the primary colors and the white reference are:

Table 1.11: sRGB Chromaticities

Color	x	y
Red	0.640	0.330
Green	0.300	0.600
Blue	0.150	0.060
White Reference (D65)	0.3127	0.3290

These chromaticities are identical to the Rec. 709 colorspace.

Transfer function. Note that negative values for L are only used by the Y'CbCr conversion.

$$L' = -1.055(-L)^{\frac{1}{2.4}} + 0.055, \text{ for } L < -0.0031308$$
$$L' = 12.92L, \text{ for } -0.0031308 \leq L \leq 0.0031308$$
$$L' = 1.055L^{\frac{1}{2.4}} - 0.055, \text{ for } 0.0031308 < L \leq 1$$

Inverse Transfer function:

$$L = -((-L' + 0.055)/1.055)^{2.4}, \text{ for } L' < -0.04045$$
$$L = L'/12.92, \text{ for } -0.04045 \leq L' \leq 0.04045$$
$$L = ((L' + 0.055)/1.055)^{2.4}, \text{ for } L' > 0.04045$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding as defined by *sYCC* :

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. This transform is identical to one defined in SMPTE 170M/BT.601. The Y'CbCr quantization is full range.

Colorspace Adobe RGB (V4L2_COLORSPACE_ADOBERGB)

The *AdobeRGB* standard defines the colorspace used by computer graphics that use the AdobeRGB colorspace. This is also known as the *opRGB* standard. The default transfer function is V4L2_XFER_FUNC_ADOBERGB. The default Y'CbCr encoding is V4L2_YCBCR_ENC_601. The default Y'CbCr quantization is full range. The chromaticities of the primary colors and the white reference are:

Table 1.12: Adobe RGB Chromaticities

Color	x	y
Red	0.6400	0.3300
Green	0.2100	0.7100
Blue	0.1500	0.0600
White Reference (D65)	0.3127	0.3290

Transfer function:

$$L' = L^{\frac{1}{2.19921875}}$$

Inverse Transfer function:

$$L = L'^{(2.19921875)}$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. This transform is identical to one defined in SMPTE 170M/BT.601. The Y'CbCr quantization is full range.

Colorspace BT.2020 (V4L2_COLORSPACE_BT2020)

The *ITU BT.2020* standard defines the colorspace used by Ultra-high definition television (UHDTV). The default transfer function is V4L2_XFER_FUNC_709. The default Y'CbCr encoding is V4L2_YCBCR_ENC_BT2020. The default R'G'B' quantization is limited range (!), and so is the default Y'CbCr quantization. The chromaticities of the primary colors and the white reference are:

Table 1.13: BT.2020 Chromaticities

Color	x	y
Red	0.708	0.292
Green	0.170	0.797
Blue	0.131	0.046
White Reference (D65)	0.3127	0.3290

Transfer function (same as Rec. 709):

$$L' = 4.5L \text{, for } 0 \leq L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099 \text{, for } 0.018 \leq L \leq 1$$

Inverse Transfer function:

$$L = L'/4.5 \text{, for } L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}} \text{, for } L' \geq 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_BT2020 encoding:

$$Y' = 0.2627R' + 0.6780G' + 0.0593B'$$
$$Cb = -0.1396R' - 0.3604G' + 0.5B'$$
$$Cr = 0.5R' - 0.4598G' - 0.0402B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. The Y'CbCr quantization is limited range.

There is also an alternate constant luminance R'G'B' to Yc'CbcCrc (V4L2_YCBCR_ENC_BT2020_CONST_LUM) encoding:

Luma:

$$Yc' = (0.2627R + 0.6780G + 0.0593B)'$$
$$B' - Yc' \leq 0 :$$
$$Cbc = (B' - Yc')/1.9404$$
$$B' - Yc' > 0 :$$
$$Cbc = (B' - Yc')/1.5816$$
$$R' - Yc' \leq 0 :$$
$$Crc = (R' - Y')/1.7184$$
$$R' - Yc' > 0 :$$
$$Crc = (R' - Y')/0.9936$$

Yc' is clamped to the range [0…1] and Cbc and Crc are clamped to the range [-0.5…0.5]. The Yc'CbcCrc quantization is limited range.

Colorspace DCI-P3 (V4L2_COLORSPACE_DCI_P3)

The *SMPTE RP 431-2* standard defines the colorspace used by cinema projectors that use the DCI-P3 colorspace. The default transfer function is `V4L2_XFER_FUNC_DCI_P3`. The default Y'CbCr encoding is `V4L2_YCBCR_ENC_709`. The default Y'CbCr quantization is limited range.

Note:

Note that this colorspace standard does not specify a Y'CbCr encoding since it is not meant to be encoded to Y'CbCr. So this default Y'CbCr encoding was picked because it is the HDTV encoding.

The chromaticities of the primary colors and the white reference are:

Table 1.14: DCI-P3 Chromaticities

Color	x	y
Red	0.6800	0.3200
Green	0.2650	0.6900
Blue	0.1500	0.0600
White Reference	0.3140	0.3510

Transfer function:

$$L' = L^{\frac{1}{2.6}}$$

Inverse Transfer function:

$$L = L'^{(2.6)}$$

Y'CbCr encoding is not specified. V4L2 defaults to Rec. 709.

Colorspace SMPTE 240M (V4L2_COLORSPACE_SMPTE240M)

The *SMPTE 240M* standard was an interim standard used during the early days of HDTV (1988-1998). It has been superseded by Rec. 709. The default transfer function is `V4L2_XFER_FUNC_SMPTE240M`. The default Y'CbCr encoding is `V4L2_YCBCR_ENC_SMPTE240M`. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 1.15: SMPTE 240M Chromaticities

Color	x	y
Red	0.630	0.340
Green	0.310	0.595
Blue	0.155	0.070
White Reference (D65)	0.3127	0.3290

These chromaticities are identical to the SMPTE 170M colorspace.

Transfer function:

$$L' = 4L, \text{ for } 0 \leq L < 0.0228$$
$$L' = 1.1115L^{0.45} - 0.1115, \text{ for } 0.0228 \leq L \leq 1$$

Inverse Transfer function:

$$L = \frac{L'}{4}, \text{ for } 0 \le L' < 0.0913$$

$$L = \left(\frac{L' + 0.1115}{1.1115}\right)^{\frac{1}{0.45}}, \text{ for } L' \ge 0.0913$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_SMPTE240M encoding:

$$Y' = 0.2122R' + 0.7013G' + 0.0865B'$$
$$Cb = -0.1161R' - 0.3839G' + 0.5B'$$
$$Cr = 0.5R' - 0.4451G' - 0.0549B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. The Y'CbCr quantization is limited range.

Colorspace NTSC 1953 (V4L2_COLORSPACE_470_SYSTEM_M)

This standard defines the colorspace used by NTSC in 1953. In practice this colorspace is obsolete and SMPTE 170M should be used instead. The default transfer function is V4L2_XFER_FUNC_709. The default Y'CbCr encoding is V4L2_YCBCR_ENC_601. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 1.16: NTSC 1953 Chromaticities

Color	x	y
Red	0.67	0.33
Green	0.21	0.71
Blue	0.14	0.08
White Reference (C)	0.310	0.316

Note:

This colorspace uses Illuminant C instead of D65 as the white reference. To correctly convert an image in this colorspace to another that uses D65 you need to apply a chromatic adaptation algorithm such as the Bradford method.

The transfer function was never properly defined for NTSC 1953. The Rec. 709 transfer function is recommended in the literature:

$$L' = 4.5L, \text{ for } 0 \le L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099, \text{ for } 0.018 \le L \le 1$$

Inverse Transfer function:

$$L = \frac{L'}{4.5}, \text{ for } L' < 0.081$$

$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \ge 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. The Y'CbCr quantization is limited range. This transform is identical to one defined in SMPTE 170M/BT.601.

Colorspace EBU Tech. 3213 (V4L2_COLORSPACE_470_SYSTEM_BG)

The *EBU Tech 3213* standard defines the colorspace used by PAL/SECAM in 1975. In practice this colorspace is obsolete and SMPTE 170M should be used instead. The default transfer function is V4L2_XFER_FUNC_709. The default Y'CbCr encoding is V4L2_YCBCR_ENC_601. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 1.17: EBU Tech. 3213 Chromaticities

Color	x	y
Red	0.64	0.33
Green	0.29	0.60
Blue	0.15	0.06
White Reference (D65)	0.3127	0.3290

The transfer function was never properly defined for this colorspace. The Rec. 709 transfer function is recommended in the literature:

$$L' = 4.5L, \text{ for } 0 \leq L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099, \text{ for } 0.018 \leq L \leq 1$$

Inverse Transfer function:

$$L = \frac{L'}{4.5}, \text{ for } L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. The Y'CbCr quantization is limited range. This transform is identical to one defined in SMPTE 170M/BT.601.

Colorspace JPEG (V4L2_COLORSPACE_JPEG)

This colorspace defines the colorspace used by most (Motion-)JPEG formats. The chromaticities of the primary colors and the white reference are identical to sRGB. The transfer function use is V4L2_XFER_FUNC_SRGB. The Y'CbCr encoding is V4L2_YCBCR_ENC_601 with full range quantization where Y' is scaled to [0…255] and Cb/Cr are scaled to [-128…128] and then clipped to [-128…127].

Note:

The JPEG standard does not actually store colorspace information. So if something other than sRGB is used, then the driver will have to set that information explicitly. Effectively V4L2_COLORSPACE_JPEG can be considered to be an abbreviation for V4L2_COLORSPACE_SRGB, V4L2_YCBCR_ENC_601 and V4L2_QUANTIZATION_FULL_RANGE.

Detailed Transfer Function Descriptions

Transfer Function SMPTE 2084 (V4L2_XFER_FUNC_SMPTE2084)

The *SMPTE ST 2084* standard defines the transfer function used by High Dynamic Range content.

Constants: m1 = (2610 / 4096) / 4

m2 = (2523 / 4096) * 128

c1 = 3424 / 4096

c2 = (2413 / 4096) * 32

c3 = (2392 / 4096) * 32

Transfer function: $L' = ((c1 + c2 * L^{m1}) / (1 + c3 * L^{m1}))^{m2}$

Inverse Transfer function: $L = (\max(L'^{1/m2} - c1, 0) / (c2 - c3 * L'^{1/m2}))^{1/m1}$

Indexed Format

In this format each pixel is represented by an 8 bit index into a 256 entry ARGB palette. It is intended for *Video Output Overlays* only. There are no ioctls to access the palette, this must be done with ioctls of the Linux framebuffer API.

Table 1.18: Indexed Image Format

Identifier	Code		Byte 0							
		Bit	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_PAL8	'PAL8'		i_7	i_6	i_5	i_4	i_3	i_2	i_1	i_0

RGB Formats

Packed RGB formats

Description

These formats are designed to match the pixel formats of typical PC graphics frame buffers. They occupy 8, 16, 24 or 32 bits per pixel. These are all packed-pixel formats, meaning all the data for a pixel lie next to each other in memory.

Table 1.19: Packed RGB Image Formats

Identifier	Code		Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_RGB332	'RGB1'		r_2	r_1	r_0	g_2	g_1	g_0	b_1	b_0																									
V4L2_PIX_FMT_ARGB444	'AR12'		g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0	a_3	a_2	a_1	a_0	r_3	r_2	r_1	r_0																	
V4L2_PIX_FMT_XRGB444	'XR12'		g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0					r_3	r_2	r_1	r_0																	
V4L2_PIX_FMT_ARGB555	'AR15'		g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0	a	r_4	r_3	r_2	r_1	r_0	g_4	g_3																	
V4L2_PIX_FMT_XRGB555	'XR15'		g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0		r_4	r_3	r_2	r_1	r_0	g_4	g_3																	
V4L2_PIX_FMT_RGB565	'RGBP'		g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0	r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3																	
V4L2_PIX_FMT_ARGB555X	'AR15' | (1 << 31)		a	r_4	r_3	r_2	r_1	r_0	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0																	
V4L2_PIX_FMT_XRGB555X	'XR15' | (1 << 31)			r_4	r_3	r_2	r_1	r_0	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0																	
V4L2_PIX_FMT_RGB565X	'RGBR'		r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0																	
V4L2_PIX_FMT_BGR24	'BGR3'		b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0									
V4L2_PIX_FMT_RGB24	'RGB3'		r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0									
V4L2_PIX_FMT_BGR666	'BGRH'		b_5	b_4	b_3	b_2	b_1	b_0	g_5	g_4	g_3	g_2	g_1	g_0	r_5	r_4	r_3	r_2	r_1	r_0															
V4L2_PIX_FMT_ABGR32	'AR24'		b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
V4L2_PIX_FMT_XBGR32	'XR24'		b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0									
V4L2_PIX_FMT_ARGB32	'BA24'		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
V4L2_PIX_FMT_XRGB32	'BX24'										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Note:

Bit 7 is the most significant bit.

The usage and value of the alpha bits (a) in the ARGB and ABGR formats (collectively referred to as alpha formats) depend on the device type and hardware operation. *Capture* devices (including capture queues of mem-to-mem devices) fill the alpha component in memory. When the device outputs an alpha channel the alpha component will have a meaningful value. Otherwise, when the device doesn't output an alpha channel but can set the alpha bit to a user-configurable value, the *V4L2_CID_ALPHA_COMPONENT* control is used to specify that alpha value, and the alpha component of all pixels will be set to the value specified by that control. Otherwise a corresponding format without an alpha component (XRGB or XBGR) must be used instead of an alpha format.

Output devices (including output queues of mem-to-mem devices and *video output overlay* devices) read the alpha component from memory. When the device processes the alpha channel the alpha component must be filled with meaningful values by applications. Otherwise a corresponding format without an alpha component (XRGB or XBGR) must be used instead of an alpha format.

The XRGB and XBGR formats contain undefined bits (-). Applications, devices and drivers must ignore those bits, for both *Video Capture Interface* and *Video Output Interface* devices.

Byte Order. Each cell is one byte.

Table 1.20: RGB byte order

start + 0:	B_{00}	G_{00}	R_{00}	B_{01}	G_{01}	R_{01}	B_{02}	G_{02}	R_{02}	B_{03}	G_{03}	R_{03}
start + 12:	B_{10}	G_{10}	R_{10}	B_{11}	G_{11}	R_{11}	B_{12}	G_{12}	R_{12}	B_{13}	G_{13}	R_{13}
start + 24:	B_{20}	G_{20}	R_{20}	B_{21}	G_{21}	R_{21}	B_{22}	G_{22}	R_{22}	B_{23}	G_{23}	R_{23}
start + 36:	B_{30}	G_{30}	R_{30}	B_{31}	G_{31}	R_{31}	B_{32}	G_{32}	R_{32}	B_{33}	G_{33}	R_{33}

Formats defined in *Deprecated Packed RGB Image Formats* are deprecated and must not be used by new drivers. They are documented here for reference. The meaning of their alpha bits (a) is ill-defined and interpreted as in either the corresponding ARGB or XRGB format, depending on the driver.

Table 1.21: Deprecated Packed RGB Image Formats

Identifier	Code		Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_RGB444	'R444'		g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0	a_3	a_2	a_1	a_0	r_3	r_2	r_1	r_0																		
V4L2_PIX_FMT_RGB555	'RGBO'		g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0	a	r_4	r_3	r_2	r_1	r_0	g_4	g_3																		
V4L2_PIX_FMT_RGB555X	'RGBQ'		a	r_4	r_3	r_2	r_1	r_0	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0																		
V4L2_PIX_FMT_BGR32	'BGR4'		b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
V4L2_PIX_FMT_RGB32	'RGB4'		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

A test utility to determine which RGB formats a driver actually supports is available from the LinuxTV v4l-dvb repository. See https://linuxtv.org/repo/ for access instructions.

V4L2_PIX_FMT_SRGGB8 ('RGGB'), V4L2_PIX_FMT_SGRBG8 ('GRBG'), V4L2_PIX_FMT_SGBRG8 ('GBRG'), V4L2_PIX_FMT_SBGGR8 ('BA81'),

8-bit Bayer formats

Description

These four pixel formats are raw sRGB / Bayer formats with 8 bits per sample. Each sample is stored in a byte. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR8 image:

Byte Order. Each cell is one byte.

start + 0:	B_{00}	G_{01}	B_{02}	G_{03}
start + 4:	G_{10}	R_{11}	G_{12}	R_{13}
start + 8:	B_{20}	G_{21}	B_{22}	G_{23}
start + 12:	G_{30}	R_{31}	G_{32}	R_{33}

V4L2_PIX_FMT_SRGGB10 ('RG10'), V4L2_PIX_FMT_SGRBG10 ('BA10'), V4L2_PIX_FMT_SGBRG10 ('GB10'), V4L2_PIX_FMT_SBGGR10 ('BG10'),

V4L2_PIX_FMT_SGRBG10 V4L2_PIX_FMT_SGBRG10 V4L2_PIX_FMT_SBGGR10 10-bit Bayer formats expanded to 16 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 10 bits per sample. Each sample is stored in a 16-bit word, with 6 unused high bits filled with zeros. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of one of these formats:

Byte Order. Each cell is one byte, the 6 most significant bits in the high bytes are 0.

start + 0:	B_{00low}	B_{00high}	G_{01low}	G_{01high}	B_{02low}	B_{02high}	G_{03low}	G_{03high}
start + 8:	G_{10low}	G_{10high}	R_{11low}	R_{11high}	G_{12low}	G_{12high}	R_{13low}	R_{13high}
start + 16:	B_{20low}	B_{20high}	G_{21low}	G_{21high}	B_{22low}	B_{22high}	G_{23low}	G_{23high}
start + 24:	G_{30low}	G_{30high}	R_{31low}	R_{31high}	G_{32low}	G_{32high}	R_{33low}	R_{33high}

V4L2_PIX_FMT_SRGGB10P ('pRAA'), V4L2_PIX_FMT_SGRBG10P ('pgAA'), V4L2_PIX_FMT_SGBRG10P ('pGAA'), V4L2_PIX_FMT_SBGGR10P ('pBAA'),

V4L2_PIX_FMT_SGRBG10P V4L2_PIX_FMT_SGBRG10P V4L2_PIX_FMT_SBGGR10P 10-bit packed Bayer formats

Description

These four pixel formats are packed raw sRGB / Bayer formats with 10 bits per sample. Every four consecutive samples are packed into 5 bytes. Each of the first 4 bytes contain the 8 high order bits of the pixels, and the 5th byte contains the 2 least significants bits of each pixel, in the same order.

Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating green-red and green-blue rows. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR10P image:

Byte Order. Each cell is one byte.

start + 0:	B_{00high}	G_{01high}	B_{02high}	G_{03high}	G_{03low}(bits 7–6) B_{02low}(bits 5–4) G_{01low}(bits 3–2) B_{00low}(bits 1–0)
start + 5:	G_{10high}	R_{11high}	G_{12high}	R_{13high}	R_{13low}(bits 7–6) G_{12low}(bits 5–4) R_{11low}(bits 3–2) G_{10low}(bits 1–0)
start + 10:	B_{20high}	G_{21high}	B_{22high}	G_{23high}	G_{23low}(bits 7–6) B_{22low}(bits 5–4) G_{21low}(bits 3–2) B_{20low}(bits 1–0)
start + 15:	G_{30high}	R_{31high}	G_{32high}	R_{33high}	R_{33low}(bits 7–6) G_{32low}(bits 5–4) R_{31low}(bits 3–2) G_{30low}(bits 1–0)

V4L2_PIX_FMT_SBGGR10ALAW8 ('aBA8'), V4L2_PIX_FMT_SGBRG10ALAW8 ('aGA8'), V4L2_PIX_FMT_SGRBG10ALAW8 ('agA8'), V4L2_PIX_FMT_SRGGB10ALAW8 ('aRA8'),

V4L2_PIX_FMT_SGBRG10ALAW8 V4L2_PIX_FMT_SGRBG10ALAW8 V4L2_PIX_FMT_SRGGB10ALAW8 10-bit Bayer formats compressed to 8 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 10 bits per color compressed to 8 bits each, using the A-LAW algorithm. Each color component consumes 8 bits of memory. In other respects this format is similar to *V4L2_PIX_FMT_SRGGB8 ('RGGB'), V4L2_PIX_FMT_SGRBG8 ('GRBG'), V4L2_PIX_FMT_SGBRG8 ('GBRG'), V4L2_PIX_FMT_SBGGR8 ('BA81'),* .

V4L2_PIX_FMT_SBGGR10DPCM8 ('bBA8'), V4L2_PIX_FMT_SGBRG10DPCM8 ('bGA8'), V4L2_PIX_FMT_SGRBG10DPCM8 ('BD10'), V4L2_PIX_FMT_SRGGB10DPCM8 ('bRA8'),

man V4L2_PIX_FMT_SBGGR10DPCM8(2)

V4L2_PIX_FMT_SGBRG10DPCM8 V4L2_PIX_FMT_SGRBG10DPCM8 V4L2_PIX_FMT_SRGGB10DPCM8 10-bit Bayer formats compressed to 8 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 10 bits per colour compressed to 8 bits each, using DPCM compression. DPCM, differential pulse-code modulation, is lossy. Each colour component consumes 8 bits of memory. In other respects this format is similar to *V4L2_PIX_FMT_SRGGB10 ('RG10'), V4L2_PIX_FMT_SGRBG10 ('BA10'), V4L2_PIX_FMT_SGBRG10 ('GB10'), V4L2_PIX_FMT_SBGGR10 ('BG10'),* .

V4L2_PIX_FMT_SRGGB12 ('RG12'), V4L2_PIX_FMT_SGRBG12 ('BA12'), V4L2_PIX_FMT_SGBRG12 ('GB12'), V4L2_PIX_FMT_SBGGR12 ('BG12'),

V4L2_PIX_FMT_SGRBG12 V4L2_PIX_FMT_SGBRG12 V4L2_PIX_FMT_SBGGR12 12-bit Bayer formats expanded to 16 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 12 bits per colour. Each colour component is stored in a 16-bit word, with 4 unused high bits filled with zeros. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR12 image:

Byte Order. Each cell is one byte, the 4 most significant bits in the high bytes are 0.

start + 0:	B_{00low}	B_{00high}	G_{01low}	G_{01high}	B_{02low}	B_{02high}	G_{03low}	G_{03high}
start + 8:	G_{10low}	G_{10high}	R_{11low}	R_{11high}	G_{12low}	G_{12high}	R_{13low}	R_{13high}
start + 16:	B_{20low}	B_{20high}	G_{21low}	G_{21high}	B_{22low}	B_{22high}	G_{23low}	G_{23high}
start + 24:	G_{30low}	G_{30high}	R_{31low}	R_{31high}	G_{32low}	G_{32high}	R_{33low}	R_{33high}

V4L2_PIX_FMT_SRGGB16 ('RG16'), V4L2_PIX_FMT_SGRBG16 ('GR16'), V4L2_PIX_FMT_SGBRG16 ('GB16'), V4L2_PIX_FMT_SBGGR16 ('BYR2'),

16-bit Bayer formats

Description

These four pixel formats are raw sRGB / Bayer formats with 16 bits per sample. Each sample is stored in a 16-bit word. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating

red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR16 image:

Byte Order. Each cell is one byte.

start + 0:	B_{00low}	B_{00high}	G_{01low}	G_{01high}	B_{02low}	B_{02high}	G_{03low}	G_{03high}
start + 8:	G_{10low}	G_{10high}	R_{11low}	R_{11high}	G_{12low}	G_{12high}	R_{13low}	R_{13high}
start + 16:	B_{20low}	B_{20high}	G_{21low}	G_{21high}	B_{22low}	B_{22high}	G_{23low}	G_{23high}
start + 24:	G_{30low}	G_{30high}	R_{31low}	R_{31high}	G_{32low}	G_{32high}	R_{33low}	R_{33high}

YUV Formats

YUV is the format native to TV broadcast and composite video signals. It separates the brightness information (Y) from the color information (U and V or Cb and Cr). The color information consists of red and blue *color difference* signals, this way the green component can be reconstructed by subtracting from the brightness component. See *Colorspaces* for conversion examples. YUV was chosen because early television would only transmit brightness information. To add color in a way compatible with existing receivers a new signal carrier was added to transmit the color difference signals. Secondary in the YUV format the U and V components usually have lower resolution than the Y component. This is an analog video compression technique taking advantage of a property of the human visual system, being more sensitive to brightness information.

Packed YUV formats

Description

Similar to the packed RGB formats these formats store the Y, Cb and Cr component of each pixel in one 16 or 32 bit word.

Table 1.22: Packed YUV Image Formats

Identifier	Code		Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_YUV444	'Y444'		Cb_3	Cb_2	Cb_1	Cb_0	Cr_3	Cr_2	Cr_1	Cr_0	a_3	a_2	a_1	a_0	Y'_3	Y'_2	Y'_1	Y'_0																		
V4L2_PIX_FMT_YUV555	'YUVO'		Cb_2	Cb_1	Cb_0	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0	a	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0	Cb_4	Cb_3																		
V4L2_PIX_FMT_YUV565	'YUVP'		Cb_2	Cb_1	Cb_0	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0	Cb_5	Cb_4	Cb_3																		
V4L2_PIX_FMT_YUV32	'YUV4'		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Y'_7	Y'_6	Y'_5	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0	Cb_7	Cb_6	Cb_5	Cb_4	Cb_3	Cb_2	Cb_1	Cb_0	Cr_7	Cr_6	Cr_5	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0

Note:

1. *Bit 7 is the most significant bit;*
2. *The value of a = alpha bits is undefined when reading from the driver, ignored when writing to the driver, except when alpha blending has been negotiated for a* Video Overlay *or* Video Output Overlay *.*

V4L2_PIX_FMT_GREY ('GREY')

Grey-scale image

Description

This is a grey-scale image. It is really a degenerate Y'CbCr format which simply contains no Cb or Cr data.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}

V4L2_PIX_FMT_Y10 ('Y10 ')

Grey-scale image

Description

This is a grey-scale image with a depth of 10 bits per pixel. Pixels are stored in 16-bit words with unused high bits padded with 0. The least significant byte is stored at lower memory addresses (little-endian).

Byte Order. Each cell is one byte.

start + 0:	Y'_{00low}	Y'_{00high}	Y'_{01low}	Y'_{01high}	Y'_{02low}	Y'_{02high}	Y'_{03low}	Y'_{03high}
start + 8:	Y'_{10low}	Y'_{10high}	Y'_{11low}	Y'_{11high}	Y'_{12low}	Y'_{12high}	Y'_{13low}	Y'_{13high}
start + 16:	Y'_{20low}	Y'_{20high}	Y'_{21low}	Y'_{21high}	Y'_{22low}	Y'_{22high}	Y'_{23low}	Y'_{23high}
start + 24:	Y'_{30low}	Y'_{30high}	Y'_{31low}	Y'_{31high}	Y'_{32low}	Y'_{32high}	Y'_{33low}	Y'_{33high}

V4L2_PIX_FMT_Y12 ('Y12 ')

Grey-scale image

Description

This is a grey-scale image with a depth of 12 bits per pixel. Pixels are stored in 16-bit words with unused high bits padded with 0. The least significant byte is stored at lower memory addresses (little-endian).

Byte Order. Each cell is one byte.

start + 0:	Y'_{00low}	Y'_{00high}	Y'_{01low}	Y'_{01high}	Y'_{02low}	Y'_{02high}	Y'_{03low}	Y'_{03high}
start + 8:	Y'_{10low}	Y'_{10high}	Y'_{11low}	Y'_{11high}	Y'_{12low}	Y'_{12high}	Y'_{13low}	Y'_{13high}
start + 16:	Y'_{20low}	Y'_{20high}	Y'_{21low}	Y'_{21high}	Y'_{22low}	Y'_{22high}	Y'_{23low}	Y'_{23high}
start + 24:	Y'_{30low}	Y'_{30high}	Y'_{31low}	Y'_{31high}	Y'_{32low}	Y'_{32high}	Y'_{33low}	Y'_{33high}

V4L2_PIX_FMT_Y10BPACK ('Y10B')

Grey-scale image as a bit-packed array

Description

This is a packed grey-scale image format with a depth of 10 bits per pixel. Pixels are stored in a bit-packed array of 10bit bits per pixel, with no padding between them and with the most significant bits coming first from the left.

Bit-packed representation.

pixels cross the byte boundary and have a ratio of 5 bytes for each 4 pixels.

$Y'_{00[9:2]}$	$Y'_{00[1:0]}Y'_{01[9:4]}$	$Y'_{01[3:0]}Y'_{02[9:6]}$	$Y'_{02[5:0]}Y'_{03[9:8]}$	$Y'_{03[7:0]}$

V4L2_PIX_FMT_Y16 ('Y16 ')

Grey-scale image

Description

This is a grey-scale image with a depth of 16 bits per pixel. The least significant byte is stored at lower memory addresses (little-endian).

Note:

The actual sampling precision may be lower than 16 bits, for example 10 bits per pixel with values in range 0 to 1023.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00low}	Y'_{00high}	Y'_{01low}	Y'_{01high}	Y'_{02low}	Y'_{02high}	Y'_{03low}	Y'_{03high}
start + 8:	Y'_{10low}	Y'_{10high}	Y'_{11low}	Y'_{11high}	Y'_{12low}	Y'_{12high}	Y'_{13low}	Y'_{13high}
start + 16:	Y'_{20low}	Y'_{20high}	Y'_{21low}	Y'_{21high}	Y'_{22low}	Y'_{22high}	Y'_{23low}	Y'_{23high}
start + 24:	Y'_{30low}	Y'_{30high}	Y'_{31low}	Y'_{31high}	Y'_{32low}	Y'_{32high}	Y'_{33low}	Y'_{33high}

V4L2_PIX_FMT_Y16_BE ('Y16 ' | (1 << 31))

Grey-scale image

Description

This is a grey-scale image with a depth of 16 bits per pixel. The most significant byte is stored at lower memory addresses (big-endian).

Note:

The actual sampling precision may be lower than 16 bits, for example 10 bits per pixel with values in range 0 to 1023.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00high}	Y'_{00low}	Y'_{01high}	Y'_{01low}	Y'_{02high}	Y'_{02low}	Y'_{03high}	Y'_{03low}
start + 8:	Y'_{10high}	Y'_{10low}	Y'_{11high}	Y'_{11low}	Y'_{12high}	Y'_{12low}	Y'_{13high}	Y'_{13low}
start + 16:	Y'_{20high}	Y'_{20low}	Y'_{21high}	Y'_{21low}	Y'_{22high}	Y'_{22low}	Y'_{23high}	Y'_{23low}
start + 24:	Y'_{30high}	Y'_{30low}	Y'_{31high}	Y'_{31low}	Y'_{32high}	Y'_{32low}	Y'_{33high}	Y'_{33low}

V4L2_PIX_FMT_Y8I ('Y8I ')

Interleaved grey-scale image, e.g. from a stereo-pair

Description

This is a grey-scale image with a depth of 8 bits per pixel, but with pixels from 2 sources interleaved. Each pixel is stored in a 16-bit word. E.g. the R200 RealSense camera stores pixel from the left sensor in lower and from the right sensor in the higher 8 bits.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00left}	$Y'_{00right}$	Y'_{01left}	$Y'_{01right}$	Y'_{02left}	$Y'_{02right}$	Y'_{03left}	$Y'_{03right}$
start + 8:	Y'_{10left}	$Y'_{10right}$	Y'_{11left}	$Y'_{11right}$	Y'_{12left}	$Y'_{12right}$	Y'_{13left}	$Y'_{13right}$
start + 16:	Y'_{20left}	$Y'_{20right}$	Y'_{21left}	$Y'_{21right}$	Y'_{22left}	$Y'_{22right}$	Y'_{23left}	$Y'_{23right}$
start + 24:	Y'_{30left}	$Y'_{30right}$	Y'_{31left}	$Y'_{31right}$	Y'_{32left}	$Y'_{32right}$	Y'_{33left}	$Y'_{33right}$

V4L2_PIX_FMT_Y12I ('Y12I')

Interleaved grey-scale image, e.g. from a stereo-pair

Description

This is a grey-scale image with a depth of 12 bits per pixel, but with pixels from 2 sources interleaved and bit-packed. Each pixel is stored in a 24-bit word in the little-endian order. On a little-endian machine these pixels can be deinterlaced using

```
__u8 *buf;
left0 = 0xfff & *(__u16 *)buf;
right0 = *(__u16 *)(buf + 1) >> 4;
```

Bit-packed representation. pixels cross the byte boundary and have a ratio of 3 bytes for each interleaved pixel.

$Y'_{0left[7:0]}$	$Y'_{0right[3:0]}Y'_{0left[11:8]}$	$Y'_{0right[11:4]}$

V4L2_PIX_FMT_UV8 ('UV8')

UV plane interleaved

Description

In this format there is no Y plane, Only CbCr plane. ie (UV interleaved)

Byte Order. Each cell is one byte.

start + 0:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 4:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}
start + 8:	Cb_{20}	Cr_{20}	Cb_{21}	Cr_{21}
start + 12:	Cb_{30}	Cr_{30}	Cb_{31}	Cr_{31}

V4L2_PIX_FMT_YUYV ('YUYV')

Packed format with ½ horizontal chroma resolution, also known as YUV 4:2:2

Description

In this format each four bytes is two pixels. Each four bytes is two Y's, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. As you can see, the Cr and Cb components have half the horizontal resolution of the Y component. V4L2_PIX_FMT_YUYV is known in the Windows environment as YUY2.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Cb_{00}	Y'_{01}	Cr_{00}	Y'_{02}	Cb_{01}	Y'_{03}	Cr_{01}
start + 8:	Y'_{10}	Cb_{10}	Y'_{11}	Cr_{10}	Y'_{12}	Cb_{11}	Y'_{13}	Cr_{11}
start + 16:	Y'_{20}	Cb_{20}	Y'_{21}	Cr_{20}	Y'_{22}	Cb_{21}	Y'_{23}	Cr_{21}
start + 24:	Y'_{30}	Cb_{30}	Y'_{31}	Cr_{30}	Y'_{32}	Cb_{31}	Y'_{33}	Cr_{31}

Color Sample Location..

	0		1		2		3
0	Y	C	Y		Y	C	Y
1	Y	C	Y		Y	C	Y
2	Y	C	Y		Y	C	Y
3	Y	C	Y		Y	C	Y

V4L2_PIX_FMT_UYVY ('UYVY')

Variation of `V4L2_PIX_FMT_YUYV` with different order of samples in memory

Description

In this format each four bytes is two pixels. Each four bytes is two Y's, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. As you can see, the Cr and Cb components have half the horizontal resolution of the Y component.

Byte Order. Each cell is one byte.

start + 0:	Cb_{00}	Y'_{00}	Cr_{00}	Y'_{01}	Cb_{01}	Y'_{02}	Cr_{01}	Y'_{03}
start + 8:	Cb_{10}	Y'_{10}	Cr_{10}	Y'_{11}	Cb_{11}	Y'_{12}	Cr_{11}	Y'_{13}
start + 16:	Cb_{20}	Y'_{20}	Cr_{20}	Y'_{21}	Cb_{21}	Y'_{22}	Cr_{21}	Y'_{23}
start + 24:	Cb_{30}	Y'_{30}	Cr_{30}	Y'_{31}	Cb_{31}	Y'_{32}	Cr_{31}	Y'_{33}

Color Sample Location..

	0		1	2		3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_YVYU ('YVYU')

Variation of `V4L2_PIX_FMT_YUYV` with different order of samples in memory

Description

In this format each four bytes is two pixels. Each four bytes is two Y's, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. As you can see, the Cr and Cb components have half the horizontal resolution of the Y component.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Cr_{00}	Y'_{01}	Cb_{00}	Y'_{02}	Cr_{01}	Y'_{03}	Cb_{01}
start + 8:	Y'_{10}	Cr_{10}	Y'_{11}	Cb_{10}	Y'_{12}	Cr_{11}	Y'_{13}	Cb_{11}
start + 16:	Y'_{20}	Cr_{20}	Y'_{21}	Cb_{20}	Y'_{22}	Cr_{21}	Y'_{23}	Cb_{21}
start + 24:	Y'_{30}	Cr_{30}	Y'_{31}	Cb_{30}	Y'_{32}	Cr_{31}	Y'_{33}	Cb_{31}

Color Sample Location..

	0		1	2		3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_VYUY ('VYUY')

Variation of V4L2_PIX_FMT_YUYV with different order of samples in memory

Description

In this format each four bytes is two pixels. Each four bytes is two Y's, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. As you can see, the Cr and Cb components have half the horizontal resolution of the Y component.

Byte Order. Each cell is one byte.

start + 0:	Cr_{00}	Y'_{00}	Cb_{00}	Y'_{01}	Cr_{01}	Y'_{02}	Cb_{01}	Y'_{03}
start + 8:	Cr_{10}	Y'_{10}	Cb_{10}	Y'_{11}	Cr_{11}	Y'_{12}	Cb_{11}	Y'_{13}
start + 16:	Cr_{20}	Y'_{20}	Cb_{20}	Y'_{21}	Cr_{21}	Y'_{22}	Cb_{21}	Y'_{23}
start + 24:	Cr_{30}	Y'_{30}	Cb_{30}	Y'_{31}	Cr_{31}	Y'_{32}	Cb_{31}	Y'_{33}

Color Sample Location..

	0		1		2	3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_Y41P ('Y41P')

Format with ¼ horizontal chroma resolution, also known as YUV 4:1:1

Description

In this format each 12 bytes is eight pixels. In the twelve bytes are two CbCr pairs and eight Y's. The first CbCr pair goes with the first four Y's, and the second CbCr pair goes with the other four Y's. The Cb and Cr components have one fourth the horizontal resolution of the Y component.

Do not confuse this format with *V4L2_PIX_FMT_YUV411P* . Y41P is derived from "YUV 4:1:1 *packed*", while YUV411P stands for "YUV 4:1:1 *planar*".

Byte Order. Each cell is one byte.

start + 0:	Cb_{00}	Y'_{00}	Cr_{00}	Y'_{01}	Cb_{01}	Y'_{02}	Cr_{01}	Y'_{03}	Y'_{04}	Y'_{05}	Y'_{06}	Y'_{07}
start + 12:	Cb_{10}	Y'_{10}	Cr_{10}	Y'_{11}	Cb_{11}	Y'_{12}	Cr_{11}	Y'_{13}	Y'_{14}	Y'_{15}	Y'_{16}	Y'_{17}
start + 24:	Cb_{20}	Y'_{20}	Cr_{20}	Y'_{21}	Cb_{21}	Y'_{22}	Cr_{21}	Y'_{23}	Y'_{24}	Y'_{25}	Y'_{26}	Y'_{27}
start + 36:	Cb_{30}	Y'_{30}	Cr_{30}	Y'_{31}	Cb_{31}	Y'_{32}	Cr_{31}	Y'_{33}	Y'_{34}	Y'_{35}	Y'_{36}	Y'_{37}

Color Sample Location..

	0	1		2	3	4	5		6	7
0	Y	Y	C	Y	Y	Y	Y	C	Y	Y
1	Y	Y	C	Y	Y	Y	Y	C	Y	Y
2	Y	Y	C	Y	Y	Y	Y	C	Y	Y
3	Y	Y	C	Y	Y	Y	Y	C	Y	Y

V4L2_PIX_FMT_YVU420 ('YV12'), V4L2_PIX_FMT_YUV420 ('YU12')

V4L2_PIX_FMT_YUV420 Planar formats with ½ horizontal and vertical chroma resolution, also known as YUV 4:2:0

Description

These are planar formats, as opposed to a packed format. The three components are separated into three sub- images or planes. The Y plane is first. The Y plane has one byte per pixel. For V4L2_PIX_FMT_YVU420, the Cr plane immediately follows the Y plane in memory. The Cr plane is half the width and half the height of the Y plane (and of the image). Each Cr belongs to four pixels, a two-by-two square of the image. For example, Cr_0 belongs to Y'_{00}, Y'_{01}, Y'_{10}, and Y'_{11}. Following the Cr plane is the Cb plane, just like the Cr plane. V4L2_PIX_FMT_YUV420 is the same except the Cb plane comes first, then the Cr plane.

If the Y plane has pad bytes after each row, then the Cr and Cb planes have half as many pad bytes after their rows. In other words, two Cx rows (including padding) is exactly as long as one Y row (including padding).

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cr_{00}	Cr_{01}		
start + 18:	Cr_{10}	Cr_{11}		
start + 20:	Cb_{00}	Cb_{01}		
start + 22:	Cb_{10}	Cb_{11}		

Color Sample Location..

	0		1		2		3
0	Y		Y		Y		Y
		C				C	
1	Y		Y		Y		Y
2	Y		Y		Y		Y
		C				C	
3	Y		Y		Y		Y

V4L2_PIX_FMT_YUV420M ('YM12'), V4L2_PIX_FMT_YVU420M ('YM21')

V4L2_PIX_FMT_YVU420M Variation of V4L2_PIX_FMT_YUV420 and V4L2_PIX_FMT_YVU420 with planes non contiguous in memory.

Description

This is a multi-planar format, as opposed to a packed format. The three components are separated into three sub-images or planes.

The Y plane is first. The Y plane has one byte per pixel. For V4L2_PIX_FMT_YUV420M the Cb data constitutes the second plane which is half the width and half the height of the Y plane (and of the image). Each Cb belongs to four pixels, a two-by-two square of the image. For example, Cb_0 belongs to Y'_{00}, Y'_{01}, Y'_{10}, and Y'_{11}. The Cr data, just like the Cb plane, is in the third plane.

V4L2_PIX_FMT_YVU420M is the same except the Cr data is stored in the second plane and the Cb data in the third plane.

If the Y plane has pad bytes after each row, then the Cb and Cr planes have half as many pad bytes after their rows. In other words, two Cx rows (including padding) is exactly as long as one Y row (including padding).

V4L2_PIX_FMT_YUV420M and V4L2_PIX_FMT_YVU420M are intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs* .

Byte Order. Each cell is one byte.

start0 + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start0 + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start0 + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start0 + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start1 + 0:	Cb_{00}	Cb_{01}		
start1 + 2:	Cb_{10}	Cb_{11}		
start2 + 0:	Cr_{00}	Cr_{01}		
start2 + 2:	Cr_{10}	Cr_{11}		

Color Sample Location..

	0		1		2		3
0	Y		Y		Y		Y
		C				C	
1	Y		Y		Y		Y
2	Y		Y		Y		Y
		C				C	
3	Y		Y		Y		Y

V4L2_PIX_FMT_YUV422M ('YM16'), V4L2_PIX_FMT_YVU422M ('YM61')

V4L2_PIX_FMT_YVU422M Planar formats with ½ horizontal resolution, also known as YUV and YVU 4:2:2

Description

This is a multi-planar format, as opposed to a packed format. The three components are separated into three sub-images or planes.

The Y plane is first. The Y plane has one byte per pixel. For V4L2_PIX_FMT_YUV422M the Cb data constitutes the second plane which is half the width of the Y plane (and of the image). Each Cb belongs to two pixels. For example, Cb_0 belongs to Y'_{00}, Y'_{01}. The Cr data, just like the Cb plane, is in the third plane.

V4L2_PIX_FMT_YVU422M is the same except the Cr data is stored in the second plane and the Cb data in the third plane.

If the Y plane has pad bytes after each row, then the Cb and Cr planes have half as many pad bytes after their rows. In other words, two Cx rows (including padding) is exactly as long as one Y row (including padding).

V4L2_PIX_FMT_YUV422M and V4L2_PIX_FMT_YVU422M are intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs* .

Byte Order. Each cell is one byte.

start0 + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start0 + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start0 + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start0 + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start1 + 0:	Cb_{00}	Cb_{01}		
start1 + 2:	Cb_{10}	Cb_{11}		
start1 + 4:	Cb_{20}	Cb_{21}		
start1 + 6:	Cb_{30}	Cb_{31}		
start2 + 0:	Cr_{00}	Cr_{01}		
start2 + 2:	Cr_{10}	Cr_{11}		
start2 + 4:	Cr_{20}	Cr_{21}		
start2 + 6:	Cr_{30}	Cr_{31}		

Color Sample Location..

	0		1	2		3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_YUV444M ('YM24'), V4L2_PIX_FMT_YVU444M ('YM42')

V4L2_PIX_FMT_YVU444M Planar formats with full horizontal resolution, also known as YUV and YVU 4:4:4

Description

This is a multi-planar format, as opposed to a packed format. The three components are separated into three sub-images or planes.

The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_YUV444M` the Cb data constitutes the second plane which is the same width and height as the Y plane (and as the image). The Cr data, just like the Cb plane, is in the third plane.

`V4L2_PIX_FMT_YVU444M` is the same except the Cr data is stored in the second plane and the Cb data in the third plane.

If the Y plane has pad bytes after each row, then the Cb and Cr planes have the same number of pad bytes after their rows.

`V4L2_PIX_FMT_YUV444M` and `V4L2_PIX_FMT_YUV444M` are intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs* .

Byte Order. Each cell is one byte.

start0 + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start0 + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start0 + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start0 + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start1 + 0:	Cb_{00}	Cb_{01}	Cb_{02}	Cb_{03}
start1 + 4:	Cb_{10}	Cb_{11}	Cb_{12}	Cb_{13}
start1 + 8:	Cb_{20}	Cb_{21}	Cb_{22}	Cb_{23}
start1 + 12:	Cb_{20}	Cb_{21}	Cb_{32}	Cb_{33}
start2 + 0:	Cr_{00}	Cr_{01}	Cr_{02}	Cr_{03}
start2 + 4:	Cr_{10}	Cr_{11}	Cr_{12}	Cr_{13}
start2 + 8:	Cr_{20}	Cr_{21}	Cr_{22}	Cr_{23}
start2 + 12:	Cr_{30}	Cr_{31}	Cr_{32}	Cr_{33}

Color Sample Location..

	0	1	2	3
0	YC	YC	YC	YC
1	YC	YC	YC	YC
2	YC	YC	YC	YC
3	YC	YC	YC	YC

V4L2_PIX_FMT_YVU410 ('YVU9'), V4L2_PIX_FMT_YUV410 ('YUV9')

V4L2_PIX_FMT_YUV410 Planar formats with ¼ horizontal and vertical chroma resolution, also known as YUV 4:1:0

Description

These are planar formats, as opposed to a packed format. The three components are separated into three sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_YVU410`, the Cr plane immediately follows the Y plane in memory. The Cr plane is ¼ the width and ¼ the height of the Y plane (and of the image). Each Cr belongs to 16 pixels, a four-by-four square of the image. Following the Cr plane is the Cb plane, just like the Cr plane. `V4L2_PIX_FMT_YUV410` is the same, except the Cb plane comes first, then the Cr plane.

If the Y plane has pad bytes after each row, then the Cr and Cb planes have ¼ as many pad bytes after their rows. In other words, four Cx rows (including padding) are exactly as long as one Y row (including padding).

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cr_{00}			
start + 17:	Cb_{00}			

Color Sample Location..

	0		1		2		3
0	Y		Y		Y		Y
1	Y		Y		Y		Y
				C			
2	Y		Y		Y		Y
3	Y		Y		Y		Y

V4L2_PIX_FMT_YUV422P ('422P')

Format with ½ horizontal chroma resolution, also known as YUV 4:2:2. Planar layout as opposed to V4L2_PIX_FMT_YUYV

Description

This format is not commonly used. This is a planar version of the YUYV format. The three components are separated into three sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. The Cb plane immediately follows the Y plane in memory. The Cb plane is half the width of the Y plane (and of the image). Each Cb belongs to two pixels. For example, Cb_0 belongs to Y'_{00}, Y'_{01}. Following the Cb plane is the Cr plane, just like the Cb plane.

If the Y plane has pad bytes after each row, then the Cr and Cb planes have half as many pad bytes after their rows. In other words, two Cx rows (including padding) is exactly as long as one Y row (including padding).

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}	Cb_{01}		
start + 18:	Cb_{10}	Cb_{11}		
start + 20:	Cb_{20}	Cb_{21}		
start + 22:	Cb_{30}	Cb_{31}		
start + 24:	Cr_{00}	Cr_{01}		
start + 26:	Cr_{10}	Cr_{11}		
start + 28:	Cr_{20}	Cr_{21}		
start + 30:	Cr_{30}	Cr_{31}		

Color Sample Location..

	0		1	2		3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_YUV411P ('411P')

Format with ¼ horizontal chroma resolution, also known as YUV 4:1:1. Planar layout as opposed to V4L2_PIX_FMT_Y41P

Description

This format is not commonly used. This is a planar format similar to the 4:2:2 planar format except with half as many chroma. The three components are separated into three sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. The Cb plane immediately follows the Y plane in memory. The Cb plane is ¼ the width of the Y plane (and of the image). Each Cb belongs to 4 pixels all on the same row. For example, Cb_0 belongs to Y'_{00}, Y'_{01}, Y'_{02} and Y'_{03}. Following the Cb plane is the Cr plane, just like the Cb plane.

If the Y plane has pad bytes after each row, then the Cr and Cb planes have ¼ as many pad bytes after their rows. In other words, four C x rows (including padding) is exactly as long as one Y row (including padding).

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}			
start + 17:	Cb_{10}			
start + 18:	Cb_{20}			
start + 19:	Cb_{30}			
start + 20:	Cr_{00}			
start + 21:	Cr_{10}			
start + 22:	Cr_{20}			
start + 23:	Cr_{30}			

Color Sample Location..

	0	1		2	3
0	Y	Y	C	Y	Y
1	Y	Y	C	Y	Y
2	Y	Y	C	Y	Y
3	Y	Y	C	Y	Y

V4L2_PIX_FMT_NV12 ('NV12'), V4L2_PIX_FMT_NV21 ('NV21')

V4L2_PIX_FMT_NV21 Formats with ½ horizontal and vertical chroma resolution, also known as YUV 4:2:0. One luminance and one chrominance plane with alternating chroma samples as opposed to `V4L2_PIX_FMT_YVU420`

Description

These are two-plane versions of the YUV 4:2:0 format. The three components are separated into two sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_NV12`, a combined CbCr plane immediately follows the Y plane in memory. The CbCr plane is the same width, in bytes, as the Y plane (and of the image), but is half as tall in pixels. Each CbCr pair belongs to four pixels. For example, Cb_0/Cr_0 belongs to Y'_{00}, Y'_{01}, Y'_{10}, Y'_{11}. `V4L2_PIX_FMT_NV21` is the same except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

If the Y plane has pad bytes after each row, then the CbCr plane has as many pad bytes after its rows.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 20:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}

Color Sample Location..

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y

V4L2_PIX_FMT_NV12M ('NM12'), V4L2_PIX_FMT_NV21M ('NM21'), V4L2_PIX_FMT_NV12MT_16X16

V4L2_PIX_FMT_NV21M V4L2_PIX_FMT_NV12MT_16X16 Variation of V4L2_PIX_FMT_NV12 and V4L2_PIX_FMT_NV21 with planes non contiguous in memory.

Description

This is a multi-planar, two-plane version of the YUV 4:2:0 format. The three components are separated into two sub-images or planes. V4L2_PIX_FMT_NV12M differs from V4L2_PIX_FMT_NV12 in that the two planes are non-contiguous in memory, i.e. the chroma plane do not necessarily immediately follows the luma plane. The luminance data occupies the first plane. The Y plane has one byte per pixel. In the second plane there is a chrominance data with alternating chroma samples. The CbCr plane is the same width, in bytes, as the Y plane (and of the image), but is half as tall in pixels. Each CbCr pair belongs to four pixels. For example, Cb_0/Cr_0 belongs to Y'_{00}, Y'_{01}, Y'_{10}, Y'_{11}. V4L2_PIX_FMT_NV12MT_16X16 is the tiled version of V4L2_PIX_FMT_NV12M with 16x16 macroblock tiles. Here pixels are arranged in 16x16 2D tiles and tiles are arranged in linear order in memory. V4L2_PIX_FMT_NV21M is the same as V4L2_PIX_FMT_NV12M except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

V4L2_PIX_FMT_NV12M is intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs* .

If the Y plane has pad bytes after each row, then the CbCr plane has as many pad bytes after its rows.

Byte Order. Each cell is one byte.

start0 + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start0 + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start0 + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start0 + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start1 + 0:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start1 + 4:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}

Color Sample Location..

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
2	Y		Y	Y		Y
		C				C
3	Y		Y	Y		Y

V4L2_PIX_FMT_NV12MT ('TM12')

Formats with ½ horizontal and vertical chroma resolution. This format has two planes - one for luminance and one for chrominance. Chroma samples are interleaved. The difference to V4L2_PIX_FMT_NV12 is the memory layout. Pixels are grouped in macroblocks of 64x32 size. The order of macroblocks in memory is also not standard.

Description

This is the two-plane versions of the YUV 4:2:0 format where data is grouped into 64x32 macroblocks. The three components are separated into two sub-images or planes. The Y plane has one byte per pixel and pixels are grouped into 64x32 macroblocks. The CbCr plane has the same width, in bytes, as the Y plane (and the image), but is half as tall in pixels. The chroma plane is also grouped into 64x32 macroblocks.

Width of the buffer has to be aligned to the multiple of 128, and height alignment is 32. Every four adjacent buffers - two horizontally and two vertically are grouped together and are located in memory in Z or flipped Z order.

Layout of macroblocks in memory is presented in the following figure.

Fig. 1.4: V4L2_PIX_FMT_NV12MT macroblock Z shape memory layout

The requirement that width is multiple of 128 is implemented because, the Z shape cannot be cut in half horizontally. In case the vertical resolution of macroblocks is odd then the last row of macroblocks is arranged in a linear order.

In case of chroma the layout is identical. Cb and Cr samples are interleaved. Height of the buffer is aligned to 32.

Fig. 1.5: Example V4L2_PIX_FMT_NV12MT memory layout of macroblocks

Memory layout of macroblocks of V4L2_PIX_FMT_NV12MT format in most extreme case.

V4L2_PIX_FMT_NV16 ('NV16'), V4L2_PIX_FMT_NV61 ('NV61')

V4L2_PIX_FMT_NV61 Formats with ½ horizontal chroma resolution, also known as YUV 4:2:2. One luminance and one chrominance plane with alternating chroma samples as opposed to V4L2_PIX_FMT_YVU420

Description

These are two-plane versions of the YUV 4:2:2 format. The three components are separated into two sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. For V4L2_PIX_FMT_NV16, a combined CbCr plane immediately follows the Y plane in memory. The CbCr plane is the same width and

height, in bytes, as the Y plane (and of the image). Each CbCr pair belongs to two pixels. For example, Cb_0/Cr_0 belongs to Y'_{00}, Y'_{01}. V4L2_PIX_FMT_NV61 is the same except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

If the Y plane has pad bytes after each row, then the CbCr plane has as many pad bytes after its rows.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 20:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}
start + 24:	Cb_{20}	Cr_{20}	Cb_{21}	Cr_{21}
start + 28:	Cb_{30}	Cr_{30}	Cb_{31}	Cr_{31}

Color Sample Location..

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
		C			C	
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y
		C			C	

V4L2_PIX_FMT_NV16M ('NM16'), V4L2_PIX_FMT_NV61M ('NM61')

V4L2_PIX_FMT_NV61M Variation of V4L2_PIX_FMT_NV16 and V4L2_PIX_FMT_NV61 with planes non contiguous in memory.

Description

This is a multi-planar, two-plane version of the YUV 4:2:2 format. The three components are separated into two sub-images or planes. V4L2_PIX_FMT_NV16M differs from V4L2_PIX_FMT_NV16 in that the two planes are non-contiguous in memory, i.e. the chroma plane does not necessarily immediately follow the luma plane. The luminance data occupies the first plane. The Y plane has one byte per pixel. In the second plane there is chrominance data with alternating chroma samples. The CbCr plane is the same width and height, in bytes, as the Y plane. Each CbCr pair belongs to two pixels. For example, Cb_0/Cr_0 belongs to Y'_{00}, Y'_{01}. V4L2_PIX_FMT_NV61M is the same as V4L2_PIX_FMT_NV16M except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

V4L2_PIX_FMT_NV16M and V4L2_PIX_FMT_NV61M are intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs* .

Byte Order. Each cell is one byte.

start0 + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start0 + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start0 + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start0 + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start1 + 0:	Cb_{00}	Cr_{00}	Cb_{02}	Cr_{02}
start1 + 4:	Cb_{10}	Cr_{10}	Cb_{12}	Cr_{12}
start1 + 8:	Cb_{20}	Cr_{20}	Cb_{22}	Cr_{22}
start1 + 12:	Cb_{30}	Cr_{30}	Cb_{32}	Cr_{32}

Color Sample Location..

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
		C			C	
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y
		C			C	

V4L2_PIX_FMT_NV24 ('NV24'), V4L2_PIX_FMT_NV42 ('NV42')

V4L2_PIX_FMT_NV42 Formats with full horizontal and vertical chroma resolutions, also known as YUV 4:4:4. One luminance and one chrominance plane with alternating chroma samples as opposed to V4L2_PIX_FMT_YVU420

Description

These are two-plane versions of the YUV 4:4:4 format. The three components are separated into two sub-images or planes. The Y plane is first, with each Y sample stored in one byte per pixel. For V4L2_PIX_FMT_NV24, a combined CbCr plane immediately follows the Y plane in memory. The CbCr plane has the same width and height, in pixels, as the Y plane (and the image). Each line contains one CbCr pair per pixel, with each Cb and Cr sample stored in one byte. V4L2_PIX_FMT_NV42 is the same except that the Cb and Cr samples are swapped, the CrCb plane starts with a Cr sample.

If the Y plane has pad bytes after each row, then the CbCr plane has twice as many pad bytes after its rows.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}				
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}				
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}				
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}				
start + 16:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}	Cb_{02}	Cr_{02}	Cb_{03}	Cr_{03}
start + 24:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}	Cb_{12}	Cr_{12}	Cb_{13}	Cr_{13}
start + 32:	Cb_{20}	Cr_{20}	Cb_{21}	Cr_{21}	Cb_{22}	Cr_{22}	Cb_{23}	Cr_{23}
start + 40:	Cb_{30}	Cr_{30}	Cb_{31}	Cr_{31}	Cb_{32}	Cr_{32}	Cb_{33}	Cr_{33}

V4L2_PIX_FMT_M420 ('M420')

Format with ½ horizontal and vertical chroma resolution, also known as YUV 4:2:0. Hybrid plane line-interleaved layout.

Description

M420 is a YUV format with ½ horizontal and vertical chroma subsampling (YUV 4:2:0). Pixels are organized as interleaved luma and chroma planes. Two lines of luma data are followed by one line of chroma data.

The luma plane has one byte per pixel. The chroma plane contains interleaved CbCr pixels subsampled by ½ in the horizontal and vertical directions. Each CbCr pair belongs to four pixels. For example, Cb_0/Cr_0 belongs to Y'_{00}, Y'_{01}, Y'_{10}, Y'_{11}.

All line lengths are identical: if the Y lines include pad bytes so do the CbCr lines.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 16:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 20:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 24:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}

Color Sample Location..

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y

HSV Formats

These formats store the color information of the image in a geometrical representation. The colors are mapped into a cylinder, where the angle is the HUE, the height is the VALUE and the distance to the center is the SATURATION. This is a very useful format for image segmentation algorithms.

Packed HSV formats

Description

The *hue* (h) is measured in degrees, the equivalence between degrees and LSBs depends on the hsv-encoding used, see *Colorspaces* . The *saturation* (s) and the *value* (v) are measured in percentage of the cylinder: 0 being the smallest value and 255 the maximum.

The values are packed in 24 or 32 bit formats.

Table 1.23: Packed HSV Image Formats

Identifier	Code		Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_HSV32	'HSV4'										h_7	h_6	h_5	h_4	h_3	h_2	h_1	h_0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
V4L2_PIX_FMT_HSV24	'HSV3'		h_7	h_6	h_5	h_4	h_3	h_2	h_1	h_0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0								

Bit 7 is the most significant bit.

Depth Formats

Depth data provides distance to points, mapped onto the image plane

V4L2_PIX_FMT_Z16 ('Z16 ')

16-bit depth data with distance values at each pixel

Description

This is a 16-bit format, representing depth data. Each pixel is a distance to the respective point in the image coordinates. Distance unit can vary and has to be negotiated with the device separately. Each pixel is stored in a 16-bit word in the little endian byte order.

Byte Order. Each cell is one byte.

start + 0:	Z_{00low}	Z_{00high}	Z_{01low}	Z_{01high}	Z_{02low}	Z_{02high}	Z_{03low}	Z_{03high}
start + 8:	Z_{10low}	Z_{10high}	Z_{11low}	Z_{11high}	Z_{12low}	Z_{12high}	Z_{13low}	Z_{13high}
start + 16:	Z_{20low}	Z_{20high}	Z_{21low}	Z_{21high}	Z_{22low}	Z_{22high}	Z_{23low}	Z_{23high}
start + 24:	Z_{30low}	Z_{30high}	Z_{31low}	Z_{31high}	Z_{32low}	Z_{32high}	Z_{33low}	Z_{33high}

Compressed Formats

Table 1.24: Compressed Image Formats

Identifier	Code	Details
V4L2_PIX_FMT_JPEG	'JPEG'	TBD. See also *VIDIOC_G_JPEGCOMP* , *VIDIOC_S_JPEGCOMP* .
V4L2_PIX_FMT_MPEG	'MPEG'	MPEG multiplexed stream. The actual format is determined by extended control V4L2_CID_MPEG_STREAM_TYPE, see *Codec Control IDs* .
V4L2_PIX_FMT_H264	'H264'	H264 video elementary stream with start codes.
V4L2_PIX_FMT_H264_NO_SC	'AVC1'	H264 video elementary stream without start codes.
V4L2_PIX_FMT_H264_MVC	'M264'	H264 MVC video elementary stream.
V4L2_PIX_FMT_H263	'H263'	H263 video elementary stream.
V4L2_PIX_FMT_MPEG1	'MPG1'	MPEG1 video elementary stream.
V4L2_PIX_FMT_MPEG2	'MPG2'	MPEG2 video elementary stream.
V4L2_PIX_FMT_MPEG4	'MPG4'	MPEG4 video elementary stream.
V4L2_PIX_FMT_XVID	'XVID'	Xvid video elementary stream.
V4L2_PIX_FMT_VC1_ANNEX_G	'VC1G'	VC1, SMPTE 421M Annex G compliant stream.
V4L2_PIX_FMT_VC1_ANNEX_L	'VC1L'	VC1, SMPTE 421M Annex L compliant stream.
V4L2_PIX_FMT_VP8	'VP80'	VP8 video elementary stream.
V4L2_PIX_FMT_VP9	'VP90'	VP9 video elementary stream.

SDR Formats

These formats are used for *SDR* interface only.

V4L2_SDR_FMT_CU8 ('CU08')

Complex unsigned 8-bit IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 8 bit unsigned number. I value comes first and Q value after that.

Byte Order. Each cell is one byte.

start + 0:	I'_0
start + 1:	Q'_0

V4L2_SDR_FMT_CU16LE ('CU16')

Complex unsigned 16-bit little endian IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 16 bit unsigned little endian number. I value comes first and Q value after that.

Byte Order. Each cell is one byte.

start + 0:	$I'_{0[7:0]}$	$I'_{0[15:8]}$
start + 2:	$Q'_{0[7:0]}$	$Q'_{0[15:8]}$

V4L2_SDR_FMT_CS8 ('CS08')

Complex signed 8-bit IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 8 bit signed number. I value comes first and Q value after that.

Byte Order. Each cell is one byte.

start + 0:	I'_0
start + 1:	Q'_0

V4L2_SDR_FMT_CS14LE ('CS14')

Complex signed 14-bit little endian IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 14 bit signed little endian number. I value comes first and Q value after that. 14 bit value is stored in 16 bit space with unused high bits padded with 0.

Byte Order. Each cell is one byte.

start + 0:	$I'_{0[7:0]}$	$I'_{0[13:8]}$
start + 2:	$Q'_{0[7:0]}$	$Q'_{0[13:8]}$

V4L2_SDR_FMT_RU12LE ('RU12')

Real unsigned 12-bit little endian sample

Description

This format contains sequence of real number samples. Each sample is represented as a 12 bit unsigned little endian number. Sample is stored in 16 bit space with unused high bits padded with 0.

Byte Order. Each cell is one byte.

start + 0:	$I'_{0[7:0]}$	$I'_{0[11:8]}$

Touch Formats

These formats are used for *Touch Devices* interface only.

V4L2_TCH_FMT_DELTA_TD16 ('TD16')

man V4L2_TCH_FMT_DELTA_TD16(2)

16-bit signed Touch Delta

Description

This format represents delta data from a touch controller.

Delta values may range from -32768 to 32767. Typically the values will vary through a small range depending on whether the sensor is touched or not. The full value may be seen if one of the touchscreen nodes has a fault or the line is not connected.

Byte Order. Each cell is one byte.

start + 0:	D'_{00high}	D'_{00low}	D'_{01high}	D'_{01low}	D'_{02high}	D'_{02low}	D'_{03high}	D'_{03low}
start + 8:	D'_{10high}	D'_{10low}	D'_{11high}	D'_{11low}	D'_{12high}	D'_{12low}	D'_{13high}	D'_{13low}
start + 16:	D'_{20high}	D'_{20low}	D'_{21high}	D'_{21low}	D'_{22high}	D'_{22low}	D'_{23high}	D'_{23low}
start + 24:	D'_{30high}	D'_{30low}	D'_{31high}	D'_{31low}	D'_{32high}	D'_{32low}	D'_{33high}	D'_{33low}

V4L2_TCH_FMT_DELTA_TD08 ('TD08')

man V4L2_TCH_FMT_DELTA_TD08(2)

8-bit signed Touch Delta

Description

This format represents delta data from a touch controller.

Delta values may range from -128 to 127. Typically the values will vary through a small range depending on whether the sensor is touched or not. The full value may be seen if one of the touchscreen nodes has a fault or the line is not connected.

Byte Order. Each cell is one byte.

start + 0:	D'_{00}	D'_{01}	D'_{02}	D'_{03}
start + 4:	D'_{10}	D'_{11}	D'_{12}	D'_{13}
start + 8:	D'_{20}	D'_{21}	D'_{22}	D'_{23}
start + 12:	D'_{30}	D'_{31}	D'_{32}	D'_{33}

V4L2_TCH_FMT_TU16 ('TU16')

man V4L2_TCH_FMT_TU16(2)

16-bit unsigned raw touch data

Description

This format represents unsigned 16-bit data from a touch controller.

This may be used for output for raw and reference data. Values may range from 0 to 65535.

Byte Order. Each cell is one byte.

start + 0:	R'_{00high}	R'_{00low}	R'_{01high}	R'_{01low}	R'_{02high}	R'_{02low}	R'_{03high}	R'_{03low}
start + 8:	R'_{10high}	R'_{10low}	R'_{11high}	R'_{11low}	R'_{12high}	R'_{12low}	R'_{13high}	R'_{13low}
start + 16:	R'_{20high}	R'_{20low}	R'_{21high}	R'_{21low}	R'_{22high}	R'_{22low}	R'_{23high}	R'_{23low}
start + 24:	R'_{30high}	R'_{30low}	R'_{31high}	R'_{31low}	R'_{32high}	R'_{32low}	R'_{33high}	R'_{33low}

V4L2_TCH_FMT_TU08 ('TU08')

man V4L2_TCH_FMT_TU08(2)

8-bit unsigned raw touch data

Description

This format represents unsigned 8-bit data from a touch controller.

This may be used for output for raw and reference data. Values may range from 0 to 255.

Byte Order. Each cell is one byte.

start + 0:	R'_{00}	R'_{01}	R'_{02}	R'_{03}
start + 4:	R'_{10}	R'_{11}	R'_{12}	R'_{13}
start + 8:	R'_{20}	R'_{21}	R'_{22}	R'_{23}
start + 12:	R'_{30}	R'_{31}	R'_{32}	R'_{33}

Reserved Format Identifiers

These formats are not defined by this specification, they are just listed for reference and to avoid naming conflicts. If you want to register your own format, send an e-mail to the linux-media mailing list https://linuxtv.org/lists.php for inclusion in the `videodev2.h` file. If you want to share your format with other

developers add a link to your documentation and send a copy to the linux-media mailing list for inclusion in this section. If you think your format should be listed in a standard format section please make a proposal on the linux-media mailing list.

Table 1.25: Reserved Image Formats

Identifier	Code	Details
V4L2_PIX_FMT_DV	'dvsd'	unknown
V4L2_PIX_FMT_ET61X251	'E625'	Compressed format of the ET61X251 driver.
V4L2_PIX_FMT_HI240	'HI24'	8 bit RGB format used by the BTTV driver.
V4L2_PIX_FMT_HM12	'HM12'	YUV 4:2:0 format used by the IVTV driver, http://www.ivtvdriver.org/ The format is documented in the kernel sources in the file Documentation/video4linux/cx2341x/README.hm12
V4L2_PIX_FMT_CPIA1	'CPIA'	YUV format used by the gspca cpia1 driver.
V4L2_PIX_FMT_JPGL	'JPGL'	JPEG-Light format (Pegasus Lossless JPEG) used in Divio webcams NW 80x.
V4L2_PIX_FMT_SPCA501	'S501'	YUYV per line used by the gspca driver.
V4L2_PIX_FMT_SPCA505	'S505'	YYUV per line used by the gspca driver.
V4L2_PIX_FMT_SPCA508	'S508'	YUVY per line used by the gspca driver.
V4L2_PIX_FMT_SPCA561	'S561'	Compressed GBRG Bayer format used by the gspca driver.
V4L2_PIX_FMT_PAC207	'P207'	Compressed BGGR Bayer format used by the gspca driver.
V4L2_PIX_FMT_MR97310A	'M310'	Compressed BGGR Bayer format used by the gspca driver.
V4L2_PIX_FMT_JL2005BCD	'JL20'	JPEG compressed RGGB Bayer format used by the gspca driver.
V4L2_PIX_FMT_OV511	'O511'	OV511 JPEG format used by the gspca driver.
V4L2_PIX_FMT_OV518	'O518'	OV518 JPEG format used by the gspca driver.
V4L2_PIX_FMT_PJPG	'PJPG'	Pixart 73xx JPEG format used by the gspca driver.
V4L2_PIX_FMT_SE401	'S401'	Compressed RGB format used by the gspca se401 driver
V4L2_PIX_FMT_SQ905C	'905C'	Compressed RGGB bayer format used by the gspca driver.
V4L2_PIX_FMT_MJPEG	'MJPG'	Compressed format used by the Zoran driver
V4L2_PIX_FMT_PWC1	'PWC1'	Compressed format of the PWC driver.
V4L2_PIX_FMT_PWC2	'PWC2'	Compressed format of the PWC driver.
V4L2_PIX_FMT_SN9C10X	'S910'	Compressed format of the SN9C102 driver.
V4L2_PIX_FMT_SN9C20X_I420	'S920'	YUV 4:2:0 format of the gspca sn9c20x driver.
V4L2_PIX_FMT_SN9C2028	'SONX'	Compressed GBRG bayer format of the gspca sn9c2028 driver.
V4L2_PIX_FMT_STV0680	'S680'	Bayer format of the gspca stv0680 driver.
V4L2_PIX_FMT_WNVA	'WNVA'	Used by the Winnov Videum driver, http://www.thedirks.org/winnov/
V4L2_PIX_FMT_TM6000	'TM60'	Used by Trident tm6000
V4L2_PIX_FMT_CIT_YYVYUY	'CITV'	Used by xirlink CIT, found at IBM webcams. Uses one line of Y then 1 line of VYUY
V4L2_PIX_FMT_KONICA420	'KONI'	Used by Konica webcams. YUV420 planar in blocks of 256 pixels.
V4L2_PIX_FMT_YYUV	'YYUV'	unknown
V4L2_PIX_FMT_Y4	'Y04 '	Old 4-bit greyscale format. Only the most significant 4 bits of each byte are used, the other bits are set to 0.

Continued on next page

Table 1.25 – continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_Y6	'Y06 '	Old 6-bit greyscale format. Only the most significant 6 bits of each byte are used, the other bits are set to 0.
V4L2_PIX_FMT_S5C_UYVY_JPG	'S5CI'	Two-planar format used by Samsung S5C73MX cameras. The first plane contains interleaved JPEG and UYVY image data, followed by meta data in form of an array of offsets to the UYVY data blocks. The actual pointer array follows immediately the interleaved JPEG/UYVY data, the number of entries in this array equals the height of the UYVY image. Each entry is a 4-byte unsigned integer in big endian order and it's an offset to a single pixel line of the UYVY image. The first plane can start either with JPEG or UYVY data chunk. The size of a single UYVY block equals the UYVY image's width multiplied by 2. The size of a JPEG chunk depends on the image and can vary with each line. The second plane, at an offset of 4084 bytes, contains a 4-byte offset to the pointer array in the first plane. This offset is followed by a 4-byte value indicating size of the pointer array. All numbers in the second plane are also in big endian order. Remaining data in the second plane is undefined. The information in the second plane allows to easily find location of the pointer array, which can be different for each frame. The size of the pointer array is constant for given UYVY image height. In order to extract UYVY and JPEG frames an application can initially set a data pointer to the start of first plane and then add an offset from the first entry of the pointers table. Such a pointer indicates start of an UYVY image pixel line. Whole UYVY line can be copied to a separate buffer. These steps should be repeated for each line, i.e. the number of entries in the pointer array. Anything what's in between the UYVY lines is JPEG data and should be concatenated to form the JPEG stream.
V4L2_PIX_FMT_MT21C	'MT21'	Compressed two-planar YVU420 format used by Mediatek MT8173. The compression is lossless. It is an opaque intermediate format and the MDP hardware must be used to convert V4L2_PIX_FMT_MT21C to V4L2_PIX_FMT_NV12M, V4L2_PIX_FMT_YUV420M or V4L2_PIX_FMT_YVU420.

Table 1.26: Format Flags

V4L2_PIX_FMT_FLAG_PREMUL_ALPHA	0x00000001	The color values are premultiplied by the alpha channel value. For example, if a light blue pixel with 50% transparency was described by RGBA values (128, 192, 255, 128), the same pixel described with premultiplied colors would be described by RGBA values (64, 96, 128, 128)

1.2.3 Input/Output

The V4L2 API defines several different methods to read from or write to a device. All drivers exchanging data with applications must support at least one of them.

The classic I/O method using the *read()* and *write()* function is automatically selected after opening a V4L2 device. When the driver does not support this method attempts to read or write will fail at any time.

Other methods must be negotiated. To select the streaming I/O method with memory mapped or user buffers applications call the *ioctl VIDIOC_REQBUFS* ioctl. The asynchronous I/O method is not defined yet.

Video overlay can be considered another I/O method, although the application does not directly receive the image data. It is selected by initiating video overlay with the *VIDIOC_S_FMT* ioctl. For more information see *Video Overlay Interface* .

Generally exactly one I/O method, including overlay, is associated with each file descriptor. The only exceptions are applications not exchanging data with a driver ("panel applications", see *Opening and Closing Devices*) and drivers permitting simultaneous video capturing and overlay using the same file descriptor, for compatibility with V4L and earlier versions of V4L2.

VIDIOC_S_FMT and *ioctl VIDIOC_REQBUFS* would permit this to some degree, but for simplicity drivers need not support switching the I/O method (after first switching away from read/write) other than by closing and reopening the device.

The following sections describe the various I/O methods in more detail.

Read/Write

Input and output devices support the *read()* and *write()* function, respectively, when the `V4L2_CAP_READWRITE` flag in the `capabilities` field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl is set.

Drivers may need the CPU to copy the data, but they may also support DMA to or from user memory, so this I/O method is not necessarily less efficient than other methods merely exchanging buffer pointers. It is considered inferior though because no meta-information like frame counters or timestamps are passed. This information is necessary to recognize frame dropping and to synchronize with other data streams. However this is also the simplest I/O method, requiring little or no setup to exchange data. It permits command line stunts like this (the vidctrl tool is fictitious):

```
$ vidctrl /dev/video --input=0 --format=YUYV --size=352x288
$ dd if=/dev/video of=myimage.422 bs=202752 count=1
```

To read from the device applications use the *read()* function, to write the *write()* function. Drivers must implement one I/O method if they exchange data with applications, but it need not be this. [1] When reading or writing is supported, the driver must also support the *select()* and *poll()* function. [2]

Streaming I/O (Memory Mapping)

Input and output devices support this I/O method when the `V4L2_CAP_STREAMING` flag in the `capabilities` field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl is set. There are two streaming methods, to determine if the memory mapping flavor is supported applications must call the *ioctl VIDIOC_REQBUFS* ioctl with the memory type set to `V4L2_MEMORY_MMAP`.

Streaming is an I/O method where only pointers to buffers are exchanged between application and driver, the data itself is not copied. Memory mapping is primarily intended to map buffers in device memory into the application's address space. Device memory can be for example the video memory on a graphics

[1] It would be desirable if applications could depend on drivers supporting all I/O interfaces, but as much as the complex memory mapping I/O can be inadequate for some devices we have no reason to require this interface, which is most useful for simple applications capturing still images.

[2] At the driver level *select()* and *poll()* are the same, and *select()* is too important to be optional.

card with a video capture add-on. However, being the most efficient I/O method available for a long time, many other drivers support streaming as well, allocating buffers in DMA-able main memory.

A driver can support many sets of buffers. Each set is identified by a unique buffer type value. The sets are independent and each set can hold a different type of data. To access different sets at the same time different file descriptors must be used. [1]

To allocate device buffers applications call the *ioctl VIDIOC_REQBUFS* ioctl with the desired number of buffers and buffer type, for example `V4L2_BUF_TYPE_VIDEO_CAPTURE`. This ioctl can also be used to change the number of buffers or to free the allocated memory, provided none of the buffers are still mapped.

Before applications can access the buffers they must map them into their address space with the *mmap()* function. The location of the buffers in device memory can be determined with the *ioctl VIDIOC_QUERYBUF* ioctl. In the single-planar API case, the `m.offset` and `length` returned in a struct *v4l2_buffer* are passed as sixth and second parameter to the *mmap()* function. When using the multi-planar API, struct *v4l2_buffer* contains an array of struct *v4l2_plane* structures, each containing its own `m.offset` and `length`. When using the multi-planar API, every plane of every buffer has to be mapped separately, so the number of calls to *mmap()* should be equal to number of buffers times number of planes in each buffer. The offset and length values must not be modified. Remember, the buffers are allocated in physical memory, as opposed to virtual memory, which can be swapped out to disk. Applications should free the buffers as soon as possible with the *munmap()* function.

Example: Mapping buffers in the single-planar API

```
struct v4l2_requestbuffers reqbuf;
struct {
    void *start;
    size_t length;
} *buffers;
unsigned int i;

memset(&reqbuf, 0, sizeof(reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_MMAP;
reqbuf.count = 20;

if (-1 == ioctl (fd, VIDIOC_REQBUFS, &reqbuf)) {
    if (errno == EINVAL)
        printf("Video capturing or mmap-streaming is not supported\\n");
    else
        perror("VIDIOC_REQBUFS");

    exit(EXIT_FAILURE);
}

/* We want at least five buffers. */

if (reqbuf.count < 5) {
    /* You may need to free the buffers here. */
    printf("Not enough buffer memory\\n");
    exit(EXIT_FAILURE);
}

buffers = calloc(reqbuf.count, sizeof(*buffers));
assert(buffers != NULL);

for (i = 0; i < reqbuf.count; i++) {
    struct v4l2_buffer buffer;
```

[1] One could use one file descriptor and set the buffer type field accordingly when calling *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* etc., but it makes the *select()* function ambiguous. We also like the clean approach of one file descriptor per logical stream. Video overlay for example is also a logical stream, although the CPU is not needed for continuous operation.

```
    memset(&buffer, 0, sizeof(buffer));
    buffer.type = reqbuf.type;
    buffer.memory = V4L2_MEMORY_MMAP;
    buffer.index = i;

    if (-1 == ioctl (fd, VIDIOC_QUERYBUF, &buffer)) {
        perror("VIDIOC_QUERYBUF");
        exit(EXIT_FAILURE);
    }

    buffers[i].length = buffer.length; /* remember for munmap() */

    buffers[i].start = mmap(NULL, buffer.length,
                PROT_READ | PROT_WRITE, /* recommended */
                MAP_SHARED,             /* recommended */
                fd, buffer.m.offset);

    if (MAP_FAILED == buffers[i].start) {
        /* If you do not exit here you should unmap() and free()
           the buffers mapped so far. */
        perror("mmap");
        exit(EXIT_FAILURE);
    }
}

/* Cleanup. */

for (i = 0; i < reqbuf.count; i++)
    munmap(buffers[i].start, buffers[i].length);
```

Example: Mapping buffers in the multi-planar API

```
struct v4l2_requestbuffers reqbuf;
/* Our current format uses 3 planes per buffer */
#define FMT_NUM_PLANES = 3

struct {
    void *start[FMT_NUM_PLANES];
    size_t length[FMT_NUM_PLANES];
} *buffers;
unsigned int i, j;

memset(&reqbuf, 0, sizeof(reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
reqbuf.memory = V4L2_MEMORY_MMAP;
reqbuf.count = 20;

if (ioctl(fd, VIDIOC_REQBUFS, &reqbuf) < 0) {
    if (errno == EINVAL)
        printf("Video capturing or mmap-streaming is not supported\\n");
    else
        perror("VIDIOC_REQBUFS");

    exit(EXIT_FAILURE);
}

/* We want at least five buffers. */

if (reqbuf.count < 5) {
```

```
    /* You may need to free the buffers here. */
    printf("Not enough buffer memory\\n");
    exit(EXIT_FAILURE);
}

buffers = calloc(reqbuf.count, sizeof(*buffers));
assert(buffers != NULL);

for (i = 0; i < reqbuf.count; i++) {
    struct v4l2_buffer buffer;
    struct v4l2_plane planes[FMT_NUM_PLANES];

    memset(&buffer, 0, sizeof(buffer));
    buffer.type = reqbuf.type;
    buffer.memory = V4L2_MEMORY_MMAP;
    buffer.index = i;
    /* length in struct v4l2_buffer in multi-planar API stores the size
     * of planes array. */
    buffer.length = FMT_NUM_PLANES;
    buffer.m.planes = planes;

    if (ioctl(fd, VIDIOC_QUERYBUF, &buffer) < 0) {
        perror("VIDIOC_QUERYBUF");
        exit(EXIT_FAILURE);
    }

    /* Every plane has to be mapped separately */
    for (j = 0; j < FMT_NUM_PLANES; j++) {
        buffers[i].length[j] = buffer.m.planes[j].length; /* remember for munmap() */

        buffers[i].start[j] = mmap(NULL, buffer.m.planes[j].length,
                  PROT_READ | PROT_WRITE, /* recommended */
                  MAP_SHARED,             /* recommended */
                  fd, buffer.m.planes[j].m.offset);

        if (MAP_FAILED == buffers[i].start[j]) {
            /* If you do not exit here you should unmap() and free()
               the buffers and planes mapped so far. */
            perror("mmap");
            exit(EXIT_FAILURE);
        }
    }
}

/* Cleanup. */

for (i = 0; i < reqbuf.count; i++)
    for (j = 0; j < FMT_NUM_PLANES; j++)
        munmap(buffers[i].start[j], buffers[i].length[j]);
```

Conceptually streaming drivers maintain two buffer queues, an incoming and an outgoing queue. They separate the synchronous capture or output operation locked to a video clock from the application which is subject to random disk or network delays and preemption by other processes, thereby reducing the probability of data loss. The queues are organized as FIFOs, buffers will be output in the order enqueued in the incoming FIFO, and were captured in the order dequeued from the outgoing FIFO.

The driver may require a minimum number of buffers enqueued at all times to function, apart of this no limit exists on the number of buffers applications can enqueue in advance, or dequeue and process. They can also enqueue in a different order than buffers have been dequeued, and the driver can *fill* enqueued *empty* buffers in any order. [2] The index number of a buffer (struct *v4l2_buffer* index) plays no role here,

[2] Random enqueue order permits applications processing images out of order (such as video codecs) to return buffers earlier, reducing the probability of data loss. Random fill order allows drivers to reuse buffers on a LIFO-basis, taking advantage of caches

it only identifies the buffer.

Initially all mapped buffers are in dequeued state, inaccessible by the driver. For capturing applications it is customary to first enqueue all mapped buffers, then to start capturing and enter the read loop. Here the application waits until a filled buffer can be dequeued, and re-enqueues the buffer when the data is no longer needed. Output applications fill and enqueue buffers, when enough buffers are stacked up the output is started with *VIDIOC_STREAMON* . In the write loop, when the application runs out of free buffers, it must wait until an empty buffer can be dequeued and reused.

To enqueue and dequeue a buffer applications use the *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* and *VIDIOC_DQBUF* ioctl. The status of a buffer being mapped, enqueued, full or empty can be determined at any time using the *ioctl VIDIOC_QUERYBUF* ioctl. Two methods exist to suspend execution of the application until one or more buffers can be dequeued. By default *VIDIOC_DQBUF* blocks when no buffer is in the outgoing queue. When the O_NONBLOCK flag was given to the *open()* function, *VIDIOC_DQBUF* returns immediately with an EAGAIN error code when no buffer is available. The *select()* or *poll()* functions are always available.

To start and stop capturing or output applications call the *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctl.

Drivers implementing memory mapping I/O must support the *VIDIOC_REQBUFS* , *VIDIOC_QUERYBUF* , *VIDIOC_QBUF* , *VIDIOC_DQBUF* , *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctls, the *mmap()* , *munmap()* , *select()* and *poll()* function. [3]

[capture example]

Streaming I/O (User Pointers)

Input and output devices support this I/O method when the V4L2_CAP_STREAMING flag in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl is set. If the particular user pointer method (not only memory mapping) is supported must be determined by calling the *ioctl VIDIOC_REQBUFS* ioctl with the memory type set to V4L2_MEMORY_USERPTR.

This I/O method combines advantages of the read/write and memory mapping methods. Buffers (planes) are allocated by the application itself, and can reside for example in virtual or shared memory. Only pointers to data are exchanged, these pointers and meta-information are passed in struct *v4l2_buffer* (or in struct *v4l2_plane* in the multi-planar API case). The driver must be switched into user pointer I/O mode by calling the *ioctl VIDIOC_REQBUFS* with the desired buffer type. No buffers (planes) are allocated beforehand, consequently they are not indexed and cannot be queried like mapped buffers with the *VIDIOC_QUERYBUF* ioctl.

Example: Initiating streaming I/O with user pointers

```
struct v4l2_requestbuffers reqbuf;

memset (&reqbuf, 0, sizeof (reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_USERPTR;

if (ioctl (fd, VIDIOC_REQBUFS, &reqbuf) == -1) {
    if (errno == EINVAL)
        printf ("Video capturing or user pointer streaming is not supported\\n");
    else
        perror ("VIDIOC_REQBUFS");

    exit (EXIT_FAILURE);
}
```

holding scatter-gather lists and the like.

[3] At the driver level *select()* and *poll()* are the same, and *select()* is too important to be optional. The rest should be evident.

Buffer (plane) addresses and sizes are passed on the fly with the *VIDIOC_QBUF* ioctl. Although buffers are commonly cycled, applications can pass different addresses and sizes at each *VIDIOC_QBUF* call. If required by the hardware the driver swaps memory pages within physical memory to create a continuous area of memory. This happens transparently to the application in the virtual memory subsystem of the kernel. When buffer pages have been swapped out to disk they are brought back and finally locked in physical memory for DMA. [1]

Filled or displayed buffers are dequeued with the *VIDIOC_DQBUF* ioctl. The driver can unlock the memory pages at any time between the completion of the DMA and this ioctl. The memory is also unlocked when *VIDIOC_STREAMOFF* is called, *ioctl VIDIOC_REQBUFS* , or when the device is closed. Applications must take care not to free buffers without dequeuing. For once, the buffers remain locked until further, wasting physical memory. Second the driver will not be notified when the memory is returned to the application's free list and subsequently reused for other purposes, possibly completing the requested DMA and overwriting valuable data.

For capturing applications it is customary to enqueue a number of empty buffers, to start capturing and enter the read loop. Here the application waits until a filled buffer can be dequeued, and re-enqueues the buffer when the data is no longer needed. Output applications fill and enqueue buffers, when enough buffers are stacked up output is started. In the write loop, when the application runs out of free buffers it must wait until an empty buffer can be dequeued and reused. Two methods exist to suspend execution of the application until one or more buffers can be dequeued. By default *VIDIOC_DQBUF* blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the *open()* function, *VIDIOC_DQBUF* returns immediately with an `EAGAIN` error code when no buffer is available. The *select()* or *poll()* function are always available.

To start and stop capturing or output applications call the *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctl.

Note:

*ref:*VIDIOC_STREAMOFF <VIDIOC_STREAMON> *removes all buffers from both queues and unlocks all buffers as a side effect. Since there is no notion of doing anything "now" on a multitasking system, if an application needs to synchronize with another event it should examine the struct* `v4l2_buffer` *timestamp of captured or outputted buffers.*

Drivers implementing user pointer I/O must support the *VIDIOC_REQBUFS* , *VIDIOC_QBUF* , *VIDIOC_DQBUF* , *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctls, the *select()* and *poll()* function. [2]

Streaming I/O (DMA buffer importing)

The DMABUF framework provides a generic method for sharing buffers between multiple devices. Device drivers that support DMABUF can export a DMA buffer to userspace as a file descriptor (known as the exporter role), import a DMA buffer from userspace using a file descriptor previously exported for a different or the same device (known as the importer role), or both. This section describes the DMABUF importer role API in V4L2.

Refer to *DMABUF exporting* for details about exporting V4L2 buffers as DMABUF file descriptors.

Input and output devices support the streaming I/O method when the `V4L2_CAP_STREAMING` flag in the `capabilities` field of struct *v4l2_capability* returned by the *VIDIOC_QUERYCAP* ioctl is set. Whether importing DMA buffers through DMABUF file descriptors is supported is determined by calling the *VIDIOC_REQBUFS* ioctl with the memory type set to `V4L2_MEMORY_DMABUF`.

[1] We expect that frequently used buffers are typically not swapped out. Anyway, the process of swapping, locking or generating scatter-gather lists may be time consuming. The delay can be masked by the depth of the incoming buffer queue, and perhaps by maintaining caches assuming a buffer will be soon enqueued again. On the other hand, to optimize memory usage drivers can limit the number of buffers locked in advance and recycle the most recently used buffers first. Of course, the pages of empty buffers in the incoming queue need not be saved to disk. Output buffers must be saved on the incoming and outgoing queue because an application may share them with other processes.

[2] At the driver level *select()* and *poll()* are the same, and *select()* is too important to be optional. The rest should be evident.

This I/O method is dedicated to sharing DMA buffers between different devices, which may be V4L devices or other video-related devices (e.g. DRM). Buffers (planes) are allocated by a driver on behalf of an application. Next, these buffers are exported to the application as file descriptors using an API which is specific for an allocator driver. Only such file descriptor are exchanged. The descriptors and meta-information are passed in struct *v4l2_buffer* (or in struct *v4l2_plane* in the multi-planar API case). The driver must be switched into DMABUF I/O mode by calling the *VIDIOC_REQBUFS* with the desired buffer type.

Example: Initiating streaming I/O with DMABUF file descriptors

```
struct v4l2_requestbuffers reqbuf;

memset(&reqbuf, 0, sizeof (reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_DMABUF;
reqbuf.count = 1;

if (ioctl(fd, VIDIOC_REQBUFS, &reqbuf) == -1) {
    if (errno == EINVAL)
        printf("Video capturing or DMABUF streaming is not supported\\n");
    else
        perror("VIDIOC_REQBUFS");

    exit(EXIT_FAILURE);
}
```

The buffer (plane) file descriptor is passed on the fly with the *VIDIOC_QBUF* ioctl. In case of multiplanar buffers, every plane can be associated with a different DMABUF descriptor. Although buffers are commonly cycled, applications can pass a different DMABUF descriptor at each *VIDIOC_QBUF* call.

Example: Queueing DMABUF using single plane API

```
int buffer_queue(int v4lfd, int index, int dmafd)
{
    struct v4l2_buffer buf;

    memset(&buf, 0, sizeof buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_DMABUF;
    buf.index = index;
    buf.m.fd = dmafd;

    if (ioctl(v4lfd, VIDIOC_QBUF, &buf) == -1) {
        perror("VIDIOC_QBUF");
        return -1;
    }

    return 0;
}
```

Example 3.6. Queueing DMABUF using multi plane API

```
int buffer_queue_mp(int v4lfd, int index, int dmafd[], int n_planes)
{
    struct v4l2_buffer buf;
    struct v4l2_plane planes[VIDEO_MAX_PLANES];
    int i;
```

```
    memset(&buf, 0, sizeof buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
    buf.memory = V4L2_MEMORY_DMABUF;
    buf.index = index;
    buf.m.planes = planes;
    buf.length = n_planes;

    memset(&planes, 0, sizeof planes);

    for (i = 0; i < n_planes; ++i)
        buf.m.planes[i].m.fd = dmafd[i];

    if (ioctl(v4lfd, VIDIOC_QBUF, &buf) == -1) {
        perror("VIDIOC_QBUF");
        return -1;
    }

    return 0;
}
```

Captured or displayed buffers are dequeued with the *VIDIOC_DQBUF* ioctl. The driver can unlock the buffer at any time between the completion of the DMA and this ioctl. The memory is also unlocked when *VIDIOC_STREAMOFF* is called, *VIDIOC_REQBUFS* , or when the device is closed.

For capturing applications it is customary to enqueue a number of empty buffers, to start capturing and enter the read loop. Here the application waits until a filled buffer can be dequeued, and re-enqueues the buffer when the data is no longer needed. Output applications fill and enqueue buffers, when enough buffers are stacked up output is started. In the write loop, when the application runs out of free buffers it must wait until an empty buffer can be dequeued and reused. Two methods exist to suspend execution of the application until one or more buffers can be dequeued. By default *VIDIOC_DQBUF* blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the *open()* function, *VIDIOC_DQBUF* returns immediately with an `EAGAIN` error code when no buffer is available. The *select()* and *poll()* functions are always available.

To start and stop capturing or displaying applications call the *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctls.

Note:

VIDIOC_STREAMOFF *removes all buffers from both queues and unlocks all buffers as a side effect. Since there is no notion of doing anything "now" on a multitasking system, if an application needs to synchronize with another event it should examine the struct* `v4l2_buffer` *timestamp of captured or outputted buffers.*

Drivers implementing DMABUF importing I/O must support the *VIDIOC_REQBUFS* , *VIDIOC_QBUF* , *VIDIOC_DQBUF* , *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctls, and the *select()* and *poll()* functions.

Asynchronous I/O

This method is not defined yet.

Buffers

A buffer contains data exchanged by application and driver using one of the Streaming I/O methods. In the multi-planar API, the data is held in planes, while the buffer structure acts as a container for the planes. Only pointers to buffers (planes) are exchanged, the data itself is not copied. These pointers, together with meta-information like timestamps or field parity, are stored in a struct *v4l2_buffer*, argument to the *ioctl*

VIDIOC_QUERYBUF , *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* and *VIDIOC_DQBUF* ioctl. In the multi-planar API, some plane-specific members of struct *v4l2_buffer*, such as pointers and sizes for each plane, are stored in struct struct *v4l2_plane* instead. In that case, struct struct *v4l2_buffer* contains an array of plane structures.

Dequeued video buffers come with timestamps. The driver decides at which part of the frame and with which clock the timestamp is taken. Please see flags in the masks V4L2_BUF_FLAG_TIMESTAMP_MASK and V4L2_BUF_FLAG_TSTAMP_SRC_MASK in *Buffer Flags* . These flags are always valid and constant across all buffers during the whole video stream. Changes in these flags may take place as a side effect of *VIDIOC_S_INPUT* or *VIDIOC_S_OUTPUT* however. The V4L2_BUF_FLAG_TIMESTAMP_COPY timestamp type which is used by e.g. on mem-to-mem devices is an exception to the rule: the timestamp source flags are copied from the OUTPUT video buffer to the CAPTURE video buffer.

v4l2_buffer

struct v4l2_buffer

Table 1.27: struct v4l2_buffer

__u32	index		Number of the buffer, set by the application except when calling *VIDIOC_DQBUF* , then it is set by the driver. This field can range from zero to the number of buffers allocated with the *ioctl VIDIOC_REQBUFS* ioctl (struct *v4l2_requestbuffers* count), plus any buffers allocated with *ioctl VIDIOC_CREATE_BUFS* minus one.
__u32	type		Type of the buffer, same as struct *v4l2_format* type or struct *v4l2_requestbuffers* type, set by the application. See *v4l2_buf_type*
__u32	bytesused		The number of bytes occupied by the data in the buffer. It depends on the negotiated data format and may change with each buffer for compressed variable size data like JPEG images. Drivers must set this field when type refers to a capture stream, applications when it refers to an output stream. If the application sets this to 0 for an output stream, then bytesused will be set to the size of the buffer (see the length field of this struct) by the driver. For multiplanar formats this field is ignored and the planes pointer is used instead.
__u32	flags		Flags set by the application or driver, see *Buffer Flags* .
__u32	field		Indicates the field order of the image in the buffer, see *v4l2_field*. This field is not used when the buffer contains VBI data. Drivers must set it when type refers to a capture stream, applications when it refers to an output stream.
struct timeval	timestamp		For capture streams this is time when the first data byte was captured, as returned by the clock_gettime() function for the relevant clock id; see V4L2_BUF_FLAG_TIMESTAMP_* in *Buffer Flags* . For output streams the driver stores the time at which the last data byte was actually sent out in the timestamp field. This permits applications to monitor the drift between the video and system clock. For output streams that use V4L2_BUF_FLAG_TIMESTAMP_COPY the application has to fill in the timestamp which will be copied by the driver to the capture stream.

Continued on next page

Table 1.27 – continued from previous page

struct *v4l2_timecode*	timecode		When type is V4L2_BUF_TYPE_VIDEO_CAPTURE and the V4L2_BUF_FLAG_TIMECODE flag is set in flags, this structure contains a frame timecode. In *V4L2_FIELD_ALTERNATE* mode the top and bottom field contain the same timecode. Timecodes are intended to help video editing and are typically recorded on video tapes, but also embedded in compressed formats like MPEG. This field is independent of the timestamp and sequence fields.
__u32	sequence		Set by the driver, counting the frames (not fields!) in sequence. This field is set for both input and output devices.

In *V4L2_FIELD_ALTERNATE* mode the top and bottom field have the same sequence number. The count starts at zero and includes dropped or repeated frames. A dropped frame was received by an input device but could not be stored due to lack of free buffer space. A repeated frame was displayed again by an output device because the application did not pass new data in time.

Note:

This may count the frames received e.g. over USB, without taking into account the frames dropped by the remote hardware due to limited compression throughput or bus bandwidth. These devices identify by not enumerating any video standards, see Video Standards *.*

__u32	memory		This field must be set by applications and/or drivers in accordance with the selected I/O method. See *v4l2_memory*
union	m		
	__u32	offset	For the single-planar API and when memory is V4L2_MEMORY_MMAP this is the offset of the buffer from the start of the device memory. The value is returned by the driver and apart of serving as parameter to the *mmap()* function not useful for applications. See *Streaming I/O (Memory Mapping)* for details
	unsigned long	userptr	For the single-planar API and when memory is V4L2_MEMORY_USERPTR this is a pointer to the buffer (casted to unsigned long type) in virtual memory, set by the application. See *Streaming I/O (User Pointers)* for details.
	struct v4l2_plane	*planes	When using the multi-planar API, contains a userspace pointer to an array of struct *v4l2_plane*. The size of the array should be put in the length field of this struct *v4l2_buffer* structure.
	int	fd	For the single-plane API and when memory is V4L2_MEMORY_DMABUF this is the file descriptor associated with a DMABUF buffer.

Continued on next page

Table 1.27 – continued from previous page

__u32	length		Size of the buffer (not the payload) in bytes for the single-planar API. This is set by the driver based on the calls to *ioctl VIDIOC_REQBUFS* and/or *ioctl VIDIOC_CREATE_BUFS* . For the multi-planar API the application sets this to the number of elements in the planes array. The driver will fill in the actual number of valid elements in that array.
__u32	reserved2		A place holder for future extensions. Drivers and applications must set this to 0.
__u32	reserved		A place holder for future extensions. Drivers and applications must set this to 0.

v4l2_plane

struct v4l2_plane

__u32	bytesused		The number of bytes occupied by data in the plane (its payload). Drivers must set this field when type refers to a capture stream, applications when it refers to an output stream. If the application sets this to 0 for an output stream, then bytesused will be set to the size of the plane (see the length field of this struct) by the driver. ***Note:*** *Note that the actual image data starts at data_offset which may not be 0.*
__u32	length		Size in bytes of the plane (not its payload). This is set by the driver based on the calls to *ioctl VIDIOC_REQBUFS* and/or *ioctl VIDIOC_CREATE_BUFS* .
union	m		
	__u32	mem_offset	When the memory type in the containing struct *v4l2_buffer* is V4L2_MEMORY_MMAP, this is the value that should be passed to *mmap()* , similar to the offset field in struct *v4l2_buffer*.
	unsigned long	userptr	When the memory type in the containing struct *v4l2_buffer* is V4L2_MEMORY_USERPTR, this is a userspace pointer to the memory allocated for this plane by an application.
	int	fd	When the memory type in the containing struct *v4l2_buffer* is V4L2_MEMORY_DMABUF, this is a file descriptor associated with a DMABUF buffer, similar to the fd field in struct *v4l2_buffer*.

Continued on next page

Table 1.28 – continued from previous page

__u32	data_offset		Offset in bytes to video data in the plane. Drivers must set this field when type refers to a capture stream, applications when it refers to an output stream. ***Note:*** *That data_offset is included in bytesused. So the size of the image in the plane is bytesused-data_offset at offset data_offset from the start of the plane.*
__u32	reserved[11]		Reserved for future use. Should be zeroed by drivers and applications.

v4l2_buf_type

enum v4l2_buf_type

V4L2_BUF_TYPE_VIDEO_CAPTURE	1	Buffer of a single-planar video capture stream, see *Video Capture Interface* .
V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE	9	Buffer of a multi-planar video capture stream, see *Video Capture Interface* .
V4L2_BUF_TYPE_VIDEO_OUTPUT	2	Buffer of a single-planar video output stream, see *Video Output Interface* .
V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE	10	Buffer of a multi-planar video output stream, see *Video Output Interface* .
V4L2_BUF_TYPE_VIDEO_OVERLAY	3	Buffer for video overlay, see *Video Overlay Interface* .
V4L2_BUF_TYPE_VBI_CAPTURE	4	Buffer of a raw VBI capture stream, see *Raw VBI Data Interface* .
V4L2_BUF_TYPE_VBI_OUTPUT	5	Buffer of a raw VBI output stream, see *Raw VBI Data Interface* .
V4L2_BUF_TYPE_SLICED_VBI_CAPTURE	6	Buffer of a sliced VBI capture stream, see *Sliced VBI Data Interface* .
V4L2_BUF_TYPE_SLICED_VBI_OUTPUT	7	Buffer of a sliced VBI output stream, see *Sliced VBI Data Interface* .
V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY	8	Buffer for video output overlay (OSD), see *Video Output Overlay Interface* .
V4L2_BUF_TYPE_SDR_CAPTURE	11	Buffer for Software Defined Radio (SDR) capture stream, see *Software Defined Radio Interface (SDR)* .
V4L2_BUF_TYPE_SDR_OUTPUT	12	Buffer for Software Defined Radio (SDR) output stream, see *Software Defined Radio Interface (SDR)* .

Buffer Flags

V4L2_BUF_FLAG_MAPPED	0x00000001	The buffer resides in device memory and has been mapped into the application's address space, see *Streaming I/O (Memory Mapping)* for details. Drivers set or clear this flag when the *ioctl VIDIOC_QUERYBUF* , *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* or *VIDIOC_DQBUF* ioctl is called. Set by the driver.

Continued on next page

Table 1.29 – continued from previous page

V4L2_BUF_FLAG_QUEUED	0x00000002	Internally drivers maintain two buffer queues, an incoming and outgoing queue. When this flag is set, the buffer is currently on the incoming queue. It automatically moves to the outgoing queue after the buffer has been filled (capture devices) or displayed (output devices). Drivers set or clear this flag when the VIDIOC_QUERYBUF ioctl is called. After (successful) calling the VIDIOC_QBUFioctl it is always set and after VIDIOC_DQBUF always cleared.
V4L2_BUF_FLAG_DONE	0x00000004	When this flag is set, the buffer is currently on the outgoing queue, ready to be dequeued from the driver. Drivers set or clear this flag when the VIDIOC_QUERYBUF ioctl is called. After calling the VIDIOC_QBUF or VIDIOC_DQBUF it is always cleared. Of course a buffer cannot be on both queues at the same time, the V4L2_BUF_FLAG_QUEUED and V4L2_BUF_FLAG_DONE flag are mutually exclusive. They can be both cleared however, then the buffer is in "dequeued" state, in the application domain so to say.
V4L2_BUF_FLAG_ERROR	0x00000040	When this flag is set, the buffer has been dequeued successfully, although the data might have been corrupted. This is recoverable, streaming may continue as normal and the buffer may be reused normally. Drivers set this flag when the VIDIOC_DQBUF ioctl is called.
V4L2_BUF_FLAG_KEYFRAME	0x00000008	Drivers set or clear this flag when calling the VIDIOC_DQBUF ioctl. It may be set by video capture devices when the buffer contains a compressed image which is a key frame (or field), i. e. can be decompressed on its own. Also known as an I-frame. Applications can set this bit when type refers to an output stream.
V4L2_BUF_FLAG_PFRAME	0x00000010	Similar to V4L2_BUF_FLAG_KEYFRAME this flags predicted frames or fields which contain only differences to a previous key frame. Applications can set this bit when type refers to an output stream.
V4L2_BUF_FLAG_BFRAME	0x00000020	Similar to V4L2_BUF_FLAG_KEYFRAME this flags a bi-directional predicted frame or field which contains only the differences between the current frame and both the preceding and following key frames to specify its content. Applications can set this bit when type refers to an output stream.
V4L2_BUF_FLAG_TIMECODE	0x00000100	The timecode field is valid. Drivers set or clear this flag when the VIDIOC_DQBUF ioctl is called. Applications can set this bit and the corresponding timecode structure when type refers to an output stream.

Continued on next page

Table 1.29 – continued from previous page

V4L2_BUF_FLAG_PREPARED	0x00000400	The buffer has been prepared for I/O and can be queued by the application. Drivers set or clear this flag when the *ioctl VIDIOC_QUERYBUF*, *VIDIOC_PREPARE_BUF*, *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* or *VIDIOC_DQBUF* ioctl is called.
V4L2_BUF_FLAG_NO_CACHE_INVALIDATE	0x00000800	Caches do not have to be invalidated for this buffer. Typically applications shall use this flag if the data captured in the buffer is not going to be touched by the CPU, instead the buffer will, probably, be passed on to a DMA-capable hardware unit for further processing or output.
V4L2_BUF_FLAG_NO_CACHE_CLEAN	0x00001000	Caches do not have to be cleaned for this buffer. Typically applications shall use this flag for output buffers if the data in this buffer has not been created by the CPU but by some DMA-capable unit, in which case caches have not been used.
V4L2_BUF_FLAG_LAST	0x00100000	Last buffer produced by the hardware. mem2mem codec drivers set this flag on the capture queue for the last buffer when the *ioctl VIDIOC_QUERYBUF* or *VIDIOC_DQBUF* ioctl is called. Due to hardware limitations, the last buffer may be empty. In this case the driver will set the bytesused field to 0, regardless of the format. Any Any subsequent call to the *VIDIOC_DQBUF* ioctl will not block anymore, but return an EPIPE error code.
V4L2_BUF_FLAG_TIMESTAMP_MASK	0x0000e000	Mask for timestamp types below. To test the timestamp type, mask out bits not belonging to timestamp type by performing a logical and operation with buffer flags and timestamp mask.
V4L2_BUF_FLAG_TIMESTAMP_UNKNOWN	0x00000000	Unknown timestamp type. This type is used by drivers before Linux 3.9 and may be either monotonic (see below) or real-time (wall clock). Monotonic clock has been favoured in embedded systems whereas most of the drivers use the realtime clock. Either kinds of timestamps are available in user space via clock_gettime() using clock IDs CLOCK_MONOTONIC and CLOCK_REALTIME, respectively.
V4L2_BUF_FLAG_TIMESTAMP_MONOTONIC	0x00002000	The buffer timestamp has been taken from the CLOCK_MONOTONIC clock. To access the same clock outside V4L2, use clock_gettime().
V4L2_BUF_FLAG_TIMESTAMP_COPY	0x00004000	The CAPTURE buffer timestamp has been taken from the corresponding OUTPUT buffer. This flag applies only to mem2mem devices.

Continued on next page

Table 1.29 – continued from previous page

V4L2_BUF_FLAG_TSTAMP_SRC_MASK	0x00070000	Mask for timestamp sources below. The timestamp source defines the point of time the timestamp is taken in relation to the frame. Logical 'and' operation between the flags field and V4L2_BUF_FLAG_TSTAMP_SRC_MASK produces the value of the timestamp source. Applications must set the timestamp source when type refers to an output stream and V4L2_BUF_FLAG_TIMESTAMP_COPY is set.
V4L2_BUF_FLAG_TSTAMP_SRC_EOF	0x00000000	End Of Frame. The buffer timestamp has been taken when the last pixel of the frame has been received or the last pixel of the frame has been transmitted. In practice, software generated timestamps will typically be read from the clock a small amount of time after the last pixel has been received or transmitten, depending on the system and other activity in it.
V4L2_BUF_FLAG_TSTAMP_SRC_SOE	0x00010000	Start Of Exposure. The buffer timestamp has been taken when the exposure of the frame has begun. This is only valid for the V4L2_BUF_TYPE_VIDEO_CAPTURE buffer type.

v4l2_memory

enum v4l2_memory

V4L2_MEMORY_MMAP	1	The buffer is used for *memory mapping* I/O.
V4L2_MEMORY_USERPTR	2	The buffer is used for *user pointer* I/O.
V4L2_MEMORY_OVERLAY	3	[to do]
V4L2_MEMORY_DMABUF	4	The buffer is used for *DMA shared buffer* I/O.

Timecodes

The struct *v4l2_timecode* structure is designed to hold a *SMPTE 12M* or similar timecode. (struct struct timeval timestamps are stored in struct *v4l2_buffer* field timestamp.)

v4l2_timecode

struct v4l2_timecode

__u32	type	Frame rate the timecodes are based on, see *Timecode Types* .
__u32	flags	Timecode flags, see *Timecode Flags* .
__u8	frames	Frame count, 0 ... 23/24/29/49/59, depending on the type of timecode.
__u8	seconds	Seconds count, 0 ... 59. This is a binary, not BCD number.
__u8	minutes	Minutes count, 0 ... 59. This is a binary, not BCD number.
__u8	hours	Hours count, 0 ... 29. This is a binary, not BCD number.
__u8	userbits[4]	The "user group" bits from the timecode.

Timecode Types

V4L2_TC_TYPE_24FPS	1	24 frames per second, i. e. film.
V4L2_TC_TYPE_25FPS	2	25 frames per second, i. e. PAL or SECAM video.
V4L2_TC_TYPE_30FPS	3	30 frames per second, i. e. NTSC video.
V4L2_TC_TYPE_50FPS	4	
V4L2_TC_TYPE_60FPS	5	

Timecode Flags

V4L2_TC_FLAG_DROPFRAME	0x0001	Indicates “drop frame” semantics for counting frames in 29.97 fps material. When set, frame numbers 0 and 1 at the start of each minute, except minutes 0, 10, 20, 30, 40, 50 are omitted from the count.
V4L2_TC_FLAG_COLORFRAME	0x0002	The “color frame” flag.
V4L2_TC_USERBITS_field	0x000C	Field mask for the “binary group flags”.
V4L2_TC_USERBITS_USERDEFINED	0x0000	Unspecified format.
V4L2_TC_USERBITS_8BITCHARS	0x0008	8-bit ISO characters.

Field Order

We have to distinguish between progressive and interlaced video. Progressive video transmits all lines of a video image sequentially. Interlaced video divides an image into two fields, containing only the odd and even lines of the image, respectively. Alternating the so called odd and even field are transmitted, and due to a small delay between fields a cathode ray TV displays the lines interleaved, yielding the original frame. This curious technique was invented because at refresh rates similar to film the image would fade out too quickly. Transmitting fields reduces the flicker without the necessity of doubling the frame rate and with it the bandwidth required for each channel.

It is important to understand a video camera does not expose one frame at a time, merely transmitting the frames separated into fields. The fields are in fact captured at two different instances in time. An object on screen may well move between one field and the next. For applications analysing motion it is of paramount importance to recognize which field of a frame is older, the *temporal order*.

When the driver provides or accepts images field by field rather than interleaved, it is also important applications understand how the fields combine to frames. We distinguish between top (aka odd) and bottom (aka even) fields, the *spatial order*: The first line of the top field is the first line of an interlaced frame, the first line of the bottom field is the second line of that frame.

However because fields were captured one after the other, arguing whether a frame commences with the top or bottom field is pointless. Any two successive top and bottom, or bottom and top fields yield a valid frame. Only when the source was progressive to begin with, e. g. when transferring film to video, two fields may come from the same frame, creating a natural order.

Counter to intuition the top field is not necessarily the older field. Whether the older field contains the top or bottom lines is a convention determined by the video standard. Hence the distinction between temporal and spatial order of fields. The diagrams below should make this clearer.

All video capture and output devices must report the current field order. Some drivers may permit the selection of a different order, to this end applications initialize the `field` field of struct *v4l2_pix_format* before calling the *VIDIOC_S_FMT* ioctl. If this is not desired it should have the value V4L2_FIELD_ANY (0).

enum v4l2_field

v4l2_field

V4L2_FIELD_ANY	0	Applications request this field order when any one of the V4L2_FIELD_NONE, V4L2_FIELD_TOP, V4L2_FIELD_BOTTOM, or V4L2_FIELD_INTERLACED formats is acceptable. Drivers choose depending on hardware capabilities or e. g. the requested image size, and return the actual field order. Drivers must never return V4L2_FIELD_ANY. If multiple field orders are possible the driver must choose one of the possible field orders during *VIDIOC_S_FMT* or *VIDIOC_TRY_FMT* . struct *v4l2_buffer* field can never be V4L2_FIELD_ANY.
V4L2_FIELD_NONE	1	Images are in progressive format, not interlaced. The driver may also indicate this order when it cannot distinguish between V4L2_FIELD_TOP and V4L2_FIELD_BOTTOM.
V4L2_FIELD_TOP	2	Images consist of the top (aka odd) field only.
V4L2_FIELD_BOTTOM	3	Images consist of the bottom (aka even) field only. Applications may wish to prevent a device from capturing interlaced images because they will have “comb” or “feathering” artefacts around moving objects.
V4L2_FIELD_INTERLACED	4	Images contain both fields, interleaved line by line. The temporal order of the fields (whether the top or bottom field is first transmitted) depends on the current video standard. M/NTSC transmits the bottom field first, all other standards the top field first.
V4L2_FIELD_SEQ_TB	5	Images contain both fields, the top field lines are stored first in memory, immediately followed by the bottom field lines. Fields are always stored in temporal order, the older one first in memory. Image sizes refer to the frame, not fields.
V4L2_FIELD_SEQ_BT	6	Images contain both fields, the bottom field lines are stored first in memory, immediately followed by the top field lines. Fields are always stored in temporal order, the older one first in memory. Image sizes refer to the frame, not fields.
V4L2_FIELD_ALTERNATE	7	The two fields of a frame are passed in separate buffers, in temporal order, i. e. the older one first. To indicate the field parity (whether the current field is a top or bottom field) the driver or application, depending on data direction, must set struct *v4l2_buffer* field to V4L2_FIELD_TOP or V4L2_FIELD_BOTTOM. Any two successive fields pair to build a frame. If fields are successive, without any dropped fields between them (fields can drop individually), can be determined from the struct *v4l2_buffer* sequence field. This format cannot be selected when using the read/write I/O method since there is no way to communicate if a field was a top or bottom field.
V4L2_FIELD_INTERLACED_TB	8	Images contain both fields, interleaved line by line, top field first. The top field is transmitted first.
V4L2_FIELD_INTERLACED_BT	9	Images contain both fields, interleaved line by line, top field first. The bottom field is transmitted first.

Field Order, Top Field First Transmitted

Field Order, Bottom Field First Transmitted

1.2.4 Interfaces

Video Capture Interface

Video capture devices sample an analog video signal and store the digitized images in memory. Today nearly all devices can capture at full 25 or 30 frames/second. With this interface applications can control the capture process and move images from the driver into user space.

Conventionally V4L2 video capture devices are accessed through character device special files named `/dev/video` and `/dev/video0` to `/dev/video63` with major number 81 and minor numbers 0 to 63. `/dev/video` is typically a symbolic link to the preferred video device.

Note:

The same device file names are used for video output devices.

Querying Capabilities

Devices supporting the video capture interface set the `V4L2_CAP_VIDEO_CAPTURE` or `V4L2_CAP_VIDEO_CAPTURE_MPLANE` flag in the `capabilities` field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. As secondary device functions they may also support the *video overlay* (`V4L2_CAP_VIDEO_OVERLAY`) and the *raw VBI capture* (`V4L2_CAP_VBI_CAPTURE`) interface. At least one of the read/write or streaming I/O methods must be supported. Tuners and audio inputs are optional.

Supplemental Functions

Video capture devices shall support *audio input* , *Tuners and Modulators* , *controls* , *cropping and scaling* and *streaming parameter* ioctls as needed. The *video input* and *video standard* ioctls must be supported by all video capture devices.

Image Format Negotiation

The result of a capture operation is determined by cropping and image format parameters. The former select an area of the video picture to capture, the latter how images are stored in memory, i. e. in RGB or YUV format, the number of bits per pixel or width and height. Together they also define how images are scaled in the process.

As usual these parameters are *not* reset at *open()* time to permit Unix tool chains, programming a device and then reading from it as if it was a plain file. Well written V4L2 applications ensure they really get what they want, including cropping and scaling.

Cropping initialization at minimum requires to reset the parameters to defaults. An example is given in *Image Cropping, Insertion and Scaling* .

To query the current image format applications set the type field of a struct *v4l2_format* to V4L2_BUF_TYPE_VIDEO_CAPTURE or V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE and call the *VIDIOC_G_FMT* ioctl with a pointer to this structure. Drivers fill the struct *v4l2_pix_format* pix or the struct *v4l2_pix_format_mplane* pix_mp member of the fmt union.

To request different parameters applications set the type field of a struct *v4l2_format* as above and initialize all fields of the struct *v4l2_pix_format* vbi member of the fmt union, or better just modify the results of *VIDIOC_G_FMT* , and call the *VIDIOC_S_FMT* ioctl with a pointer to this structure. Drivers may adjust the parameters and finally return the actual parameters as *VIDIOC_G_FMT* does.

Like *VIDIOC_S_FMT* the *VIDIOC_TRY_FMT* ioctl can be used to learn about hardware limitations without disabling I/O or possibly time consuming hardware preparations.

The contents of struct *v4l2_pix_format* and struct *v4l2_pix_format_mplane* are discussed in *Image Formats* . See also the specification of the *VIDIOC_G_FMT* , *VIDIOC_S_FMT* and *VIDIOC_TRY_FMT* ioctls for details. Video capture devices must implement both the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl, even if *VIDIOC_S_FMT* ignores all requests and always returns default parameters as *VIDIOC_G_FMT* does. *VIDIOC_TRY_FMT* is optional.

Reading Images

A video capture device may support the : *read() function* and/or streaming (*memory mapping* or *user pointer*) I/O. See *Input/Output* for details.

Video Overlay Interface

Also known as Framebuffer Overlay or Previewing.

Video overlay devices have the ability to genlock (TV-)video into the (VGA-)video signal of a graphics card, or to store captured images directly in video memory of a graphics card, typically with clipping. This can be considerable more efficient than capturing images and displaying them by other means. In the old days when only nuclear power plants needed cooling towers this used to be the only way to put live video into a window.

Video overlay devices are accessed through the same character special files as *video capture* devices.

Note:

The default function of a /dev/video device is video capturing. The overlay function is only available after calling the VIDIOC_S_FMT *ioctl.*

The driver may support simultaneous overlay and capturing using the read/write and streaming I/O methods. If so, operation at the nominal frame rate of the video standard is not guaranteed. Frames may be directed away from overlay to capture, or one field may be used for overlay and the other for capture if the capture parameters permit this.

Applications should use different file descriptors for capturing and overlay. This must be supported by all drivers capable of simultaneous capturing and overlay. Optionally these drivers may also permit capturing and overlay with a single file descriptor for compatibility with V4L and earlier versions of V4L2. [1]

[1] A common application of two file descriptors is the XFree86 *Xv/V4L* interface driver and a V4L2 application. While the X server controls video overlay, the application can take advantage of memory mapping and DMA.

In the opinion of the designers of this API, no driver writer taking the efforts to support simultaneous capturing and overlay will restrict this ability by requiring a single file descriptor, as in V4L and earlier versions of V4L2. Making this optional means applications depending on two file descriptors need backup routines to be compatible with all drivers, which is considerable more work than using two fds in applications which do not. Also two fd's fit the general concept of one file descriptor for each logical stream. Hence as a complexity trade-off drivers *must* support two file descriptors and *may* support single fd operation.

Querying Capabilities

Devices supporting the video overlay interface set the V4L2_CAP_VIDEO_OVERLAY flag in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. The overlay I/O method specified below must be supported. Tuners and audio inputs are optional.

Supplemental Functions

Video overlay devices shall support *audio input* , *Tuners and Modulators* , *controls* , *cropping and scaling* and *streaming parameter* ioctls as needed. The *video input* and *video standard* ioctls must be supported by all video overlay devices.

Setup

Before overlay can commence applications must program the driver with frame buffer parameters, namely the address and size of the frame buffer and the image format, for example RGB 5:6:5. The *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF* ioctls are available to get and set these parameters, respectively. The *VIDIOC_S_FBUF* ioctl is privileged because it allows to set up DMA into physical memory, bypassing the memory protection mechanisms of the kernel. Only the superuser can change the frame buffer address and size. Users are not supposed to run TV applications as root or with SUID bit set. A small helper application with suitable privileges should query the graphics system and program the V4L2 driver at the appropriate time.

Some devices add the video overlay to the output signal of the graphics card. In this case the frame buffer is not modified by the video device, and the frame buffer address and pixel format are not needed by the driver. The *VIDIOC_S_FBUF* ioctl is not privileged. An application can check for this type of device by calling the *VIDIOC_G_FBUF* ioctl.

A driver may support any (or none) of five clipping/blending methods:

1. Chroma-keying displays the overlaid image only where pixels in the primary graphics surface assume a certain color.
2. A bitmap can be specified where each bit corresponds to a pixel in the overlaid image. When the bit is set, the corresponding video pixel is displayed, otherwise a pixel of the graphics surface.
3. A list of clipping rectangles can be specified. In these regions *no* video is displayed, so the graphics surface can be seen here.
4. The framebuffer has an alpha channel that can be used to clip or blend the framebuffer with the video.
5. A global alpha value can be specified to blend the framebuffer contents with video images.

When simultaneous capturing and overlay is supported and the hardware prohibits different image and frame buffer formats, the format requested first takes precedence. The attempt to capture (*VIDIOC_S_FMT*) or overlay (*VIDIOC_S_FBUF*) may fail with an EBUSY error code or return accordingly modified parameters..

Overlay Window

The overlaid image is determined by cropping and overlay window parameters. The former select an area of the video picture to capture, the latter how images are overlaid and clipped. Cropping initialization at minimum requires to reset the parameters to defaults. An example is given in *Image Cropping, Insertion and Scaling* .

The overlay window is described by a struct *v4l2_window*. It defines the size of the image, its position over the graphics surface and the clipping to be applied. To get the current parameters applications set the type field of a struct *v4l2_format* to V4L2_BUF_TYPE_VIDEO_OVERLAY and call the *VIDIOC_G_FMT* ioctl.

The driver fills the struct *v4l2_window* substructure named `win`. It is not possible to retrieve a previously programmed clipping list or bitmap.

To program the overlay window applications set the `type` field of a struct *v4l2_format* to `V4L2_BUF_TYPE_VIDEO_OVERLAY`, initialize the `win` substructure and call the *VIDIOC_S_FMT* ioctl. The driver adjusts the parameters against hardware limits and returns the actual parameters as *VIDIOC_G_FMT* does. Like *VIDIOC_S_FMT* , the *VIDIOC_TRY_FMT* ioctl can be used to learn about driver capabilities without actually changing driver state. Unlike *VIDIOC_S_FMT* this also works after the overlay has been enabled.

The scaling factor of the overlaid image is implied by the width and height given in struct *v4l2_window* and the size of the cropping rectangle. For more information see *Image Cropping, Insertion and Scaling* .

When simultaneous capturing and overlay is supported and the hardware prohibits different image and window sizes, the size requested first takes precedence. The attempt to capture or overlay as well (*VIDIOC_S_FMT*) may fail with an EBUSY error code or return accordingly modified parameters.

v4l2_window

struct v4l2_window

struct v4l2_rect w Size and position of the window relative to the top, left corner of the frame buffer defined with *VIDIOC_S_FBUF* . The window can extend the frame buffer width and height, the `x` and `y` coordinates can be negative, and it can lie completely outside the frame buffer. The driver clips the window accordingly, or if that is not possible, modifies its size and/or position.

enum v4l2_field field Applications set this field to determine which video field shall be overlaid, typically one of `V4L2_FIELD_ANY` (0), `V4L2_FIELD_TOP`, `V4L2_FIELD_BOTTOM` or `V4L2_FIELD_INTERLACED`. Drivers may have to choose a different field order and return the actual setting here.

__u32 chromakey When chroma-keying has been negotiated with *VIDIOC_S_FBUF* applications set this field to the desired pixel value for the chroma key. The format is the same as the pixel format of the framebuffer (struct *v4l2_framebuffer* `fmt.pixelformat` field), with bytes in host order. E. g. for *V4L2_PIX_FMT_BGR24* the value should be 0xRRGGBB on a little endian, 0xBBGGRR on a big endian host.

struct v4l2_clip * clips When chroma-keying has *not* been negotiated and *VIDIOC_G_FBUF* indicated this capability, applications can set this field to point to an array of clipping rectangles.

Like the window coordinates `w`, clipping rectangles are defined relative to the top, left corner of the frame buffer. However clipping rectangles must not extend the frame buffer width and height, and they must not overlap. If possible applications should merge adjacent rectangles. Whether this must create x-y or y-x bands, or the order of rectangles, is not defined. When clip lists are not supported the driver ignores this field. Its contents after calling *VIDIOC_S_FMT* are undefined.

__u32 clipcount When the application set the `clips` field, this field must contain the number of clipping rectangles in the list. When clip lists are not supported the driver ignores this field, its contents after calling *VIDIOC_S_FMT* are undefined. When clip lists are supported but no clipping is desired this field must be set to zero.

void * bitmap When chroma-keying has *not* been negotiated and *VIDIOC_G_FBUF* indicated this capability, applications can set this field to point to a clipping bit mask.

It must be of the same size as the window, `w.width` and `w.height`. Each bit corresponds to a pixel in the overlaid image, which is displayed only when the bit is *set*. Pixel coordinates translate to bits like:

```
((__u8 *) bitmap)[w.width * y + x / 8] & (1 << (x & 7))
```

where `0` ≤ `x` < `w.width` and `0` ≤ `y` <`w.height`. [2]

When a clipping bit mask is not supported the driver ignores this field, its contents after calling *VIDIOC_S_FMT* are undefined. When a bit mask is supported but no clipping is desired this field must be set

[2] Should we require `w.width` to be a multiple of eight?

to NULL.

Applications need not create a clip list or bit mask. When they pass both, or despite negotiating chroma-keying, the results are undefined. Regardless of the chosen method, the clipping abilities of the hardware may be limited in quantity or quality. The results when these limits are exceeded are undefined. [3]

__u8 global_alpha The global alpha value used to blend the framebuffer with video images, if global alpha blending has been negotiated (V4L2_FBUF_FLAG_GLOBAL_ALPHA, see *VIDIOC_S_FBUF* , *Frame Buffer Flags*).

Note:

This field was added in Linux 2.6.23, extending the structure. However the VIDIOC_[G|S|TRY]_FMT *ioctls, which take a pointer to a* v4l2_format *parent structure with padding bytes at the end, are not affected.*

v4l2_clip

struct v4l2_clip [4]

struct v4l2_rect c Coordinates of the clipping rectangle, relative to the top, left corner of the frame buffer. Only window pixels *outside* all clipping rectangles are displayed.

struct v4l2_clip * next Pointer to the next clipping rectangle, NULL when this is the last rectangle. Drivers ignore this field, it cannot be used to pass a linked list of clipping rectangles.

v4l2_rect

struct v4l2_rect

__s32 left Horizontal offset of the top, left corner of the rectangle, in pixels.

__s32 top Vertical offset of the top, left corner of the rectangle, in pixels. Offsets increase to the right and down.

__u32 width Width of the rectangle, in pixels.

__u32 height Height of the rectangle, in pixels.

Enabling Overlay

To start or stop the frame buffer overlay applications call the *ioctl VIDIOC_OVERLAY* ioctl.

Video Output Interface

Video output devices encode stills or image sequences as analog video signal. With this interface applications can control the encoding process and move images from user space to the driver.

Conventionally V4L2 video output devices are accessed through character device special files named /dev/video and /dev/video0 to /dev/video63 with major number 81 and minor numbers 0 to 63. /dev/video is typically a symbolic link to the preferred video device.

[3] When the image is written into frame buffer memory it will be undesirable if the driver clips out less pixels than expected, because the application and graphics system are not aware these regions need to be refreshed. The driver should clip out more pixels or not write the image at all.

[4] The X Window system defines “regions” which are vectors of struct BoxRec { short x1,y1,x2,y2; } with width = x2 -x1 and height = y2 -y1, so one cannot pass X11 clip lists directly.

Note:

The same device file names are used also for video capture devices.

Querying Capabilities

Devices supporting the video output interface set the V4L2_CAP_VIDEO_OUTPUT or V4L2_CAP_VIDEO_OUTPUT_MPLANE flag in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. As secondary device functions they may also support the *raw VBI output* (V4L2_CAP_VBI_OUTPUT) interface. At least one of the read/write or streaming I/O methods must be supported. Modulators and audio outputs are optional.

Supplemental Functions

Video output devices shall support *audio output* , *modulator* , *controls* , *cropping and scaling* and *streaming parameter* ioctls as needed. The *video output* and *video standard* ioctls must be supported by all video output devices.

Image Format Negotiation

The output is determined by cropping and image format parameters. The former select an area of the video picture where the image will appear, the latter how images are stored in memory, i. e. in RGB or YUV format, the number of bits per pixel or width and height. Together they also define how images are scaled in the process.

As usual these parameters are *not* reset at *open()* time to permit Unix tool chains, programming a device and then writing to it as if it was a plain file. Well written V4L2 applications ensure they really get what they want, including cropping and scaling.

Cropping initialization at minimum requires to reset the parameters to defaults. An example is given in *Image Cropping, Insertion and Scaling* .

To query the current image format applications set the type field of a struct *v4l2_format* to V4L2_BUF_TYPE_VIDEO_OUTPUT or V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE and call the *VIDIOC_G_FMT* ioctl with a pointer to this structure. Drivers fill the struct *v4l2_pix_format* pix or the struct *v4l2_pix_format_mplane* pix_mp member of the fmt union.

To request different parameters applications set the type field of a struct *v4l2_format* as above and initialize all fields of the struct *v4l2_pix_format* vbi member of the fmt union, or better just modify the results of *VIDIOC_G_FMT* , and call the *VIDIOC_S_FMT* ioctl with a pointer to this structure. Drivers may adjust the parameters and finally return the actual parameters as *VIDIOC_G_FMT* does.

Like *VIDIOC_S_FMT* the *VIDIOC_TRY_FMT* ioctl can be used to learn about hardware limitations without disabling I/O or possibly time consuming hardware preparations.

The contents of struct *v4l2_pix_format* and struct *v4l2_pix_format_mplane* are discussed in *Image Formats* . See also the specification of the *VIDIOC_G_FMT* , *VIDIOC_S_FMT* and *VIDIOC_TRY_FMT* ioctls for details. Video output devices must implement both the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl, even if *VIDIOC_S_FMT* ignores all requests and always returns default parameters as *VIDIOC_G_FMT* does. *VIDIOC_TRY_FMT* is optional.

Writing Images

A video output device may support the *write() function* and/or streaming (*memory mapping* or *user pointer*) I/O. See *Input/Output* for details.

Video Output Overlay Interface

Also known as On-Screen Display (OSD)

Some video output devices can overlay a framebuffer image onto the outgoing video signal. Applications can set up such an overlay using this interface, which borrows structures and ioctls of the *Video Overlay* interface.

The OSD function is accessible through the same character special file as the *Video Output* function.

Note:

The default function of such a /dev/video device is video capturing or output. The OSD function is only available after calling the VIDIOC_S_FMT *ioctl.*

Querying Capabilities

Devices supporting the *Video Output Overlay* interface set the V4L2_CAP_VIDEO_OUTPUT_OVERLAY flag in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl.

Framebuffer

Contrary to the *Video Overlay* interface the framebuffer is normally implemented on the TV card and not the graphics card. On Linux it is accessible as a framebuffer device (/dev/fbN). Given a V4L2 device, applications can find the corresponding framebuffer device by calling the *VIDIOC_G_FBUF* ioctl. It returns, amongst other information, the physical address of the framebuffer in the base field of struct *v4l2_framebuffer*. The framebuffer device ioctl FBIOGET_FSCREENINFO returns the same address in the smem_start field of struct struct fb_fix_screeninfo. The FBIOGET_FSCREENINFO ioctl and struct fb_fix_screeninfo are defined in the linux/fb.h header file.

The width and height of the framebuffer depends on the current video standard. A V4L2 driver may reject attempts to change the video standard (or any other ioctl which would imply a framebuffer size change) with an EBUSY error code until all applications closed the framebuffer device.

Example: Finding a framebuffer device for OSD

```
#include <linux/fb.h>

struct v4l2_framebuffer fbuf;
unsigned int i;
int fb_fd;

if (-1 == ioctl(fd, VIDIOC_G_FBUF, &fbuf)) {
    perror("VIDIOC_G_FBUF");
    exit(EXIT_FAILURE);
}

for (i = 0; i < 30; i++) {
    char dev_name[16];
    struct fb_fix_screeninfo si;

    snprintf(dev_name, sizeof(dev_name), "/dev/fb%u", i);

    fb_fd = open(dev_name, O_RDWR);
    if (-1 == fb_fd) {
        switch (errno) {
```

```
        case ENOENT: /* no such file */
        case ENXIO:  /* no driver */
            continue;

        default:
            perror("open");
            exit(EXIT_FAILURE);
        }
    }

    if (0 == ioctl(fb_fd, FBIOGET_FSCREENINFO, &si)) {
        if (si.smem_start == (unsigned long)fbuf.base)
            break;
    } else {
        /* Apparently not a framebuffer device. */
    }

    close(fb_fd);
    fb_fd = -1;
}

/* fb_fd is the file descriptor of the framebuffer device
   for the video output overlay, or -1 if no device was found. */
```

Overlay Window and Scaling

The overlay is controlled by source and target rectangles. The source rectangle selects a subsection of the framebuffer image to be overlaid, the target rectangle an area in the outgoing video signal where the image will appear. Drivers may or may not support scaling, and arbitrary sizes and positions of these rectangles. Further drivers may support any (or none) of the clipping/blending methods defined for the *Video Overlay* interface.

A struct *v4l2_window* defines the size of the source rectangle, its position in the framebuffer and the clipping/blending method to be used for the overlay. To get the current parameters applications set the `type` field of a struct *v4l2_format* to V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY and call the *VIDIOC_G_FMT* ioctl. The driver fills the struct *v4l2_window* substructure named `win`. It is not possible to retrieve a previously programmed clipping list or bitmap.

To program the source rectangle applications set the `type` field of a struct *v4l2_format* to V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY, initialize the `win` substructure and call the *VIDIOC_S_FMT* ioctl. The driver adjusts the parameters against hardware limits and returns the actual parameters as *VIDIOC_G_FMT* does. Like *VIDIOC_S_FMT* , the *VIDIOC_TRY_FMT* ioctl can be used to learn about driver capabilities without actually changing driver state. Unlike *VIDIOC_S_FMT* this also works after the overlay has been enabled.

A struct *v4l2_crop* defines the size and position of the target rectangle. The scaling factor of the overlay is implied by the width and height given in struct *v4l2_window* and struct *v4l2_crop*. The cropping API applies to *Video Output* and *Video Output Overlay* devices in the same way as to *Video Capture* and *Video Overlay* devices, merely reversing the direction of the data flow. For more information see *Image Cropping, Insertion and Scaling* .

Enabling Overlay

There is no V4L2 ioctl to enable or disable the overlay, however the framebuffer interface of the driver may support the FBIOBLANK ioctl.

Codec Interface

A V4L2 codec can compress, decompress, transform, or otherwise convert video data from one format into another format, in memory. Typically such devices are memory-to-memory devices (i.e. devices with the `V4L2_CAP_VIDEO_M2M` or `V4L2_CAP_VIDEO_M2M_MPLANE` capability set).

A memory-to-memory video node acts just like a normal video node, but it supports both output (sending frames from memory to the codec hardware) and capture (receiving the processed frames from the codec hardware into memory) stream I/O. An application will have to setup the stream I/O for both sides and finally call *VIDIOC_STREAMON* for both capture and output to start the codec.

Video compression codecs use the MPEG controls to setup their codec parameters

Note:

The MPEG controls actually support many more codecs than just MPEG. See Codec Control Reference *.*

Memory-to-memory devices function as a shared resource: you can open the video node multiple times, each application setting up their own codec properties that are local to the file handle, and each can use it independently from the others. The driver will arbitrate access to the codec and reprogram it whenever another file handler gets access. This is different from the usual video node behavior where the video properties are global to the device (i.e. changing something through one file handle is visible through another file handle).

Effect Devices Interface

Note:

This interface has been be suspended from the V4L2 API. The implementation for such effects should be done via mem2mem devices.

A V4L2 video effect device can do image effects, filtering, or combine two or more images or image streams. For example video transitions or wipes. Applications send data to be processed and receive the result data either with *read()* and *write()* functions, or through the streaming I/O mechanism.

[to do]

Raw VBI Data Interface

VBI is an abbreviation of Vertical Blanking Interval, a gap in the sequence of lines of an analog video signal. During VBI no picture information is transmitted, allowing some time while the electron beam of a cathode ray tube TV returns to the top of the screen. Using an oscilloscope you will find here the vertical synchronization pulses and short data packages ASK modulated [1] onto the video signal. These are transmissions of services such as Teletext or Closed Caption.

Subject of this interface type is raw VBI data, as sampled off a video signal, or to be added to a signal for output. The data format is similar to uncompressed video images, a number of lines times a number of samples per line, we call this a VBI image.

Conventionally V4L2 VBI devices are accessed through character device special files named `/dev/vbi` and `/dev/vbi0` to `/dev/vbi31` with major number 81 and minor numbers 224 to 255. `/dev/vbi` is typically a symbolic link to the preferred VBI device. This convention applies to both input and output devices.

[1] ASK: Amplitude-Shift Keying. A high signal level represents a ‘1’ bit, a low level a ‘0’ bit.

To address the problems of finding related video and VBI devices VBI capturing and output is also available as device function under /dev/video. To capture or output raw VBI data with these devices applications must call the *VIDIOC_S_FMT* ioctl. Accessed as /dev/vbi, raw VBI capturing or output is the default device function.

Querying Capabilities

Devices supporting the raw VBI capturing or output API set the V4L2_CAP_VBI_CAPTURE or V4L2_CAP_VBI_OUTPUT flags, respectively, in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. At least one of the read/write, streaming or asynchronous I/O methods must be supported. VBI devices may or may not have a tuner or modulator.

Supplemental Functions

VBI devices shall support *video input or output* , *tuner or modulator* , and *controls* ioctls as needed. The *video standard* ioctls provide information vital to program a VBI device, therefore must be supported.

Raw VBI Format Negotiation

Raw VBI sampling abilities can vary, in particular the sampling frequency. To properly interpret the data V4L2 specifies an ioctl to query the sampling parameters. Moreover, to allow for some flexibility applications can also suggest different parameters.

As usual these parameters are *not* reset at *open()* time to permit Unix tool chains, programming a device and then reading from it as if it was a plain file. Well written V4L2 applications should always ensure they really get what they want, requesting reasonable parameters and then checking if the actual parameters are suitable.

To query the current raw VBI capture parameters applications set the type field of a struct *v4l2_format* to V4L2_BUF_TYPE_VBI_CAPTURE or V4L2_BUF_TYPE_VBI_OUTPUT, and call the *VIDIOC_G_FMT* ioctl with a pointer to this structure. Drivers fill the struct *v4l2_vbi_format* vbi member of the fmt union.

To request different parameters applications set the type field of a struct *v4l2_format* as above and initialize all fields of the struct *v4l2_vbi_format* vbi member of the fmt union, or better just modify the results of *VIDIOC_G_FMT* , and call the *VIDIOC_S_FMT* ioctl with a pointer to this structure. Drivers return an EINVAL error code only when the given parameters are ambiguous, otherwise they modify the parameters according to the hardware capabilities and return the actual parameters. When the driver allocates resources at this point, it may return an EBUSY error code to indicate the returned parameters are valid but the required resources are currently not available. That may happen for instance when the video and VBI areas to capture would overlap, or when the driver supports multiple opens and another process already requested VBI capturing or output. Anyway, applications must expect other resource allocation points which may return EBUSY, at the *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* ioctl and the first *read()* , *write()* and *select()* calls.

VBI devices must implement both the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl, even if *VIDIOC_S_FMT* ignores all requests and always returns default parameters as *VIDIOC_G_FMT* does. *VIDIOC_TRY_FMT* is optional.

v4l2_vbi_format

Table 1.30: struct v4l2_vbi_format

__u32	sampling_rate	Samples per second, i. e. unit 1 Hz.

Continued on next page

Table 1.30 – continued from previous page

__u32	offset	Horizontal offset of the VBI image, relative to the leading edge of the line synchronization pulse and counted in samples: The first sample in the VBI image will be located offset / sampling_rate seconds following the leading edge. See also *Figure 4.1. Line synchronization* .
__u32	samples_per_line	
__u32	sample_format	Defines the sample format as in *Image Formats* , a four-character-code. [2] Usually this is V4L2_PIX_FMT_GREY, i. e. each sample consists of 8 bits with lower values oriented towards the black level. Do not assume any other correlation of values with the signal level. For example, the MSB does not necessarily indicate if the signal is 'high' or 'low' because 128 may not be the mean value of the signal. Drivers shall not convert the sample format by software.
__u32	start[2]	This is the scanning system line number associated with the first line of the VBI image, of the first and the second field respectively. See *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering* for valid values. The V4L2_VBI_ITU_525_F1_START, V4L2_VBI_ITU_525_F2_START, V4L2_VBI_ITU_625_F1_START and V4L2_VBI_ITU_625_F2_START defines give the start line numbers for each field for each 525 or 625 line format as a convenience. Don't forget that ITU line numbering starts at 1, not 0. VBI input drivers can return start values 0 if the hardware cannot reliable identify scanning lines, VBI acquisition may not require this information.
__u32	count[2]	The number of lines in the first and second field image, respectively.
Drivers should be as flexibility as possible. For example, it may be possible to extend or move the VBI capture window down to the picture area, implementing a 'full field mode' to capture data service transmissions embedded in the picture. An application can set the first or second count value to zero if no data is required from the respective field; count[1] if the scanning system is progressive, i. e. not interlaced. The corresponding start value shall be ignored by the application and driver. Anyway, drivers may not support single field capturing and return both count values non-zero. Both count values set to zero, or line numbers are outside the bounds depicted[4], or a field image covering lines of two fields, are invalid and shall not be returned by the driver. To initialize the start and count fields, applications must first determine the current video standard selection. The *v4l2_std_id* or the framelines field of struct *v4l2_standard* can be evaluated for this purpose.		
__u32	flags	See *Raw VBI Format Flags* below. Currently only drivers set flags, applications must set this field to zero.
__u32	reserved[2]	This array is reserved for future extensions. Drivers and applications must set it to zero.

[2] A few devices may be unable to sample VBI data at all but can extend the video capture window to the VBI region.
[4] The valid values ar shown at *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line*

Table 1.31: Raw VBI Format Flags

V4L2_VBI_UNSYNC	0x0001	This flag indicates hardware which does not properly distinguish between fields. Normally the VBI image stores the first field (lower scanning line numbers) first in memory. This may be a top or bottom field depending on the video standard. When this flag is set the first or second field may be stored first, however the fields are still in correct temporal order with the older field first in memory. [3]
V4L2_VBI_INTERLACED	0x0002	By default the two field images will be passed sequentially; all lines of the first field followed by all lines of the second field (compare *Field Order* V4L2_FIELD_SEQ_TB and V4L2_FIELD_SEQ_BT, whether the top or bottom field is first in memory depends on the video standard). When this flag is set, the two fields are interlaced (cf. V4L2_FIELD_INTERLACED). The first line of the first field followed by the first line of the second field, then the two second lines, and so on. Such a layout may be necessary when the hardware has been programmed to capture or output interlaced video images and is unable to separate the fields for VBI capturing at the same time. For simplicity setting this flag implies that both count values are equal and non-zero.

Fig. 1.6: **Figure 4.1. Line synchronization**

Fig. 1.7: **Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)**

Fig. 1.8: **Figure 4.3. ITU-R 625 line numbering**

Remember the VBI image format depends on the selected video standard, therefore the application must choose a new standard or query the current standard first. Attempts to read or write data ahead of format negotiation, or after switching the video standard which may invalidate the negotiated VBI parameters, should be refused by the driver. A format change during active I/O is not permitted.

Reading and writing VBI images

To assure synchronization with the field number and easier implementation, the smallest unit of data passed at a time is one frame, consisting of two fields of VBI images immediately following in memory.

The total size of a frame computes as follows:

```
(count[0] + count[1]) * samples_per_line * sample size in bytes
```

The sample size is most likely always one byte, applications must check the sample_format field though, to function properly with other drivers.

A VBI device may support *read/write* and/or streaming (*memory mapping* or *user pointer*) I/O. The latter bears the possibility of synchronizing video and VBI data by using buffer timestamps.

Remember the *VIDIOC_STREAMON* ioctl and the first *read()* , *write()* and *select()* call can be resource allocation points returning an EBUSY error code if the required hardware resources are temporarily unavailable, for example the device is already in use by another process.

numbering .

[3] Most VBI services transmit on both fields, but some have different semantics depending on the field number. These cannot be reliable decoded or encoded when V4L2_VBI_UNSYNC is set.

Sliced VBI Data Interface

VBI stands for Vertical Blanking Interval, a gap in the sequence of lines of an analog video signal. During VBI no picture information is transmitted, allowing some time while the electron beam of a cathode ray tube TV returns to the top of the screen.

Sliced VBI devices use hardware to demodulate data transmitted in the VBI. V4L2 drivers shall *not* do this by software, see also the *raw VBI interface* . The data is passed as short packets of fixed size, covering one scan line each. The number of packets per video frame is variable.

Sliced VBI capture and output devices are accessed through the same character special files as raw VBI devices. When a driver supports both interfaces, the default function of a `/dev/vbi` device is *raw* VBI capturing or output, and the sliced VBI function is only available after calling the *VIDIOC_S_FMT* ioctl as defined below. Likewise a `/dev/video` device may support the sliced VBI API, however the default function here is video capturing or output. Different file descriptors must be used to pass raw and sliced VBI data simultaneously, if this is supported by the driver.

Querying Capabilities

Devices supporting the sliced VBI capturing or output API set the `V4L2_CAP_SLICED_VBI_CAPTURE` or `V4L2_CAP_SLICED_VBI_OUTPUT` flag respectively, in the `capabilities` field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. At least one of the read/write, streaming or asynchronous *I/O methods* must be supported. Sliced VBI devices may have a tuner or modulator.

Supplemental Functions

Sliced VBI devices shall support *video input or output* and *tuner or modulator* ioctls if they have these capabilities, and they may support *User Controls* ioctls. The *video standard* ioctls provide information vital to program a sliced VBI device, therefore must be supported.

Sliced VBI Format Negotiation

To find out which data services are supported by the hardware applications can call the *VIDIOC_G_SLICED_VBI_CAP* ioctl. All drivers implementing the sliced VBI interface must support this ioctl. The results may differ from those of the *VIDIOC_S_FMT* ioctl when the number of VBI lines the hardware can capture or output per frame, or the number of services it can identify on a given line are limited. For example on PAL line 16 the hardware may be able to look for a VPS or Teletext signal, but not both at the same time.

To determine the currently selected services applications set the `type` field of struct *v4l2_format* to `V4L2_BUF_TYPE_SLICED_VBI_CAPTURE` or `V4L2_BUF_TYPE_SLICED_VBI_OUTPUT`, and the *VIDIOC_G_FMT* ioctl fills the `fmt.sliced` member, a struct *v4l2_sliced_vbi_format*.

Applications can request different parameters by initializing or modifying the `fmt.sliced` member and calling the *VIDIOC_S_FMT* ioctl with a pointer to the struct *v4l2_format* structure.

The sliced VBI API is more complicated than the raw VBI API because the hardware must be told which VBI service to expect on each scan line. Not all services may be supported by the hardware on all lines (this is especially true for VBI output where Teletext is often unsupported and other services can only be inserted in one specific line). In many cases, however, it is sufficient to just set the `service_set` field to the required services and let the driver fill the `service_lines` array according to hardware capabilities. Only if more precise control is needed should the programmer set the `service_lines` array explicitly.

The *VIDIOC_S_FMT* ioctl modifies the parameters according to hardware capabilities. When the driver allocates resources at this point, it may return an EBUSY error code if the required resources are temporarily unavailable. Other resource allocation points which may return EBUSY can be the *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* ioctl and the first *read()* , *write()* and *select()* call.

v4l2_sliced_vbi_format

struct v4l2_sliced_vbi_format

<table>
<tr><td>__u32</td><td>service_set</td><td>If service_set is non-zero when passed with VIDIOC_S_FMT or VIDIOC_TRY_FMT , the service_lines array will be filled by the driver according to the services specified in this field. For example, if service_set is initialized with V4L2_SLICED_TELETEXT_B | V4L2_SLICED_WSS_625, a driver for the cx25840 video decoder sets lines 7-22 of both fields [1] to V4L2_SLICED_TELETEXT_B and line 23 of the first field to V4L2_SLICED_WSS_625. If service_set is set to zero, then the values of service_lines will be used instead.

On return the driver sets this field to the union of all elements of the returned service_lines array. It may contain less services than requested, perhaps just one, if the hardware cannot handle more services simultaneously. It may be empty (zero) if none of the requested services are supported by the hardware.</td></tr>
<tr><td>__u16</td><td>service_lines[2][24]</td><td>Applications initialize this array with sets of data services the driver shall look for or insert on the respective scan line. Subject to hardware capabilities drivers return the requested set, a subset, which may be just a single service, or an empty set. When the hardware cannot handle multiple services on the same line the driver shall choose one. No assumptions can be made on which service the driver chooses.

Data services are defined in Sliced VBI services . Array indices map to ITU-R line numbers[2] as follows:
<table>
<tr><td>Element</td><td>525 line systems</td><td>625 line systems</td></tr>
<tr><td>service_lines[0][1]</td><td>1</td><td>1</td></tr>
<tr><td>service_lines[0][23]</td><td>23</td><td>23</td></tr>
<tr><td>service_lines[1][1]</td><td>264</td><td>314</td></tr>
<tr><td>service_lines[1][23]</td><td>286</td><td>336</td></tr>
</table>
</td></tr>
<tr><td></td><td></td><td>Drivers must set service_lines [0][0] and service_lines[1][0] to zero. The V4L2_VBI_ITU_525_F1_START, V4L2_VBI_ITU_525_F2_START, V4L2_VBI_ITU_625_F1_START and V4L2_VBI_ITU_625_F2_START defines give the start line numbers for each field for each 525 or 625 line format as a convenience. Don’t forget that ITU line numbering starts at 1, not 0.</td></tr>
<tr><td>__u32</td><td>io_size</td><td>Maximum number of bytes passed by one read() or write() call, and the buffer size in bytes for the ioctl VIDIOC_QBUF, VIDIOC_DQBUF and VIDIOC_DQBUF ioctl. Drivers set this field to the size of struct v4l2_sliced_vbi_data times the number of non-zero elements in the returned service_lines array (that is the number of lines potentially carrying data).</td></tr>
</table>

Continued on next page

Table 1.32 – continued from previous page

__u32	reserved[2]	This array is reserved for future extensions. Applications and drivers must set it to zero.

Sliced VBI services

Symbol	Value	Reference	Lines, usually	Payload
V4L2_SLICED_TELETEXT_B (Teletext System B)	0x0001	*ETS 300 706* , *ITU BT.653*	PAL/SECAM line 7-22, 320-335 (second field 7-22)	Last 42 of the 45 byte Teletext packet, that is without clock run-in and framing code, lsb first transmitted.
V4L2_SLICED_VPS	0x0400	*ETS 300 231*	PAL line 16	Byte number 3 to 15 according to Figure 9 of ETS 300 231, lsb first transmitted.
V4L2_SLICED_CAPTION_525	0x1000	*CEA 608-E*	NTSC line 21, 284 (second field 21)	Two bytes in transmission order, including parity bit, lsb first transmitted.
V4L2_SLICED_WSS_625	0x4000	*ITU BT.1119* , *EN 300 294*	PAL/SECAM line 23	Byte 0 1 msb lsb msb lsb Bit 7 6 5 4 3 2 1 0 x x 13 12 11 10 9
V4L2_SLICED_VBI_525	0x1000	Set of services applicable to 525 line systems.		
V4L2_SLICED_VBI_625	0x4401	Set of services applicable to 625 line systems.		

Drivers may return an EINVAL error code when applications attempt to read or write data without prior format negotiation, after switching the video standard (which may invalidate the negotiated VBI parameters) and after switching the video input (which may change the video standard as a side effect). The *VIDIOC_S_FMT* ioctl may return an EBUSY error code when applications attempt to change the format while i/o is in progress (between a *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* and *VIDIOC_STREAMOFF* call, and after the first *read()* or *write()* call).

Reading and writing sliced VBI data

A single *read()* or *write()* call must pass all data belonging to one video frame. That is an array of struct *v4l2_sliced_vbi_data* structures with one or more elements and a total size not exceeding io_size bytes. Likewise in streaming I/O mode one buffer of io_size bytes must contain data of one video frame. The id of unused struct *v4l2_sliced_vbi_data* elements must be zero.

v4l2_sliced_vbi_data

[1] According to *ETS 300 706* lines 6-22 of the first field and lines 5-22 of the second field may carry Teletext data.
[2] See also *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering* .

struct v4l2_sliced_vbi_data

__u32	id	A flag from *Sliced VBI services* identifying the type of data in this packet. Only a single bit must be set. When the id of a captured packet is zero, the packet is empty and the contents of other fields are undefined. Applications shall ignore empty packets. When the id of a packet for output is zero the contents of the data field are undefined and the driver must no longer insert data on the requested field and line.
__u32	field	The video field number this data has been captured from, or shall be inserted at. 0 for the first field, 1 for the second field.
__u32	line	The field (as opposed to frame) line number this data has been captured from, or shall be inserted at. See *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering* for valid values. Sliced VBI capture devices can set the line number of all packets to 0 if the hardware cannot reliably identify scan lines. The field number must always be valid.
__u32	reserved	This field is reserved for future extensions. Applications and drivers must set it to zero.
__u8	data[48]	The packet payload. See *Sliced VBI services* for the contents and number of bytes passed for each data type. The contents of padding bytes at the end of this array are undefined, drivers and applications shall ignore them.

Packets are always passed in ascending line number order, without duplicate line numbers. The *write()* function and the *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* ioctl must return an EINVAL error code when applications violate this rule. They must also return an EINVAL error code when applications pass an incorrect field or line number, or a combination of field, line and id which has not been negotiated with the *VIDIOC_G_FMT* or *VIDIOC_S_FMT* ioctl. When the line numbers are unknown the driver must pass the packets in transmitted order. The driver can insert empty packets with id set to zero anywhere in the packet array.

To assure synchronization and to distinguish from frame dropping, when a captured frame does not carry any of the requested data services drivers must pass one or more empty packets. When an application fails to pass VBI data in time for output, the driver must output the last VPS and WSS packet again, and disable the output of Closed Caption and Teletext data, or output data which is ignored by Closed Caption and Teletext decoders.

A sliced VBI device may support *read/write* and/or streaming (*memory mapping* and/or *user pointer*) I/O. The latter bears the possibility of synchronizing video and VBI data by using buffer timestamps.

Sliced VBI Data in MPEG Streams

If a device can produce an MPEG output stream, it may be capable of providing *negotiated sliced VBI services* as data embedded in the MPEG stream. Users or applications control this sliced VBI data insertion with the *V4L2_CID_MPEG_STREAM_VBI_FMT* control.

If the driver does not provide the *V4L2_CID_MPEG_STREAM_VBI_FMT* control, or only allows that control to be set to *V4L2_MPEG_STREAM_VBI_FMT_NONE* , then the device cannot embed sliced VBI data in the MPEG stream.

The *V4L2_CID_MPEG_STREAM_VBI_FMT* control does not implicitly set the device driver to capture nor cease capturing sliced VBI data. The control only indicates to embed sliced VBI data in the MPEG stream,

if an application has negotiated sliced VBI service be captured.

It may also be the case that a device can embed sliced VBI data in only certain types of MPEG streams: for example in an MPEG-2 PS but not an MPEG-2 TS. In this situation, if sliced VBI data insertion is requested, the sliced VBI data will be embedded in MPEG stream types when supported, and silently omitted from MPEG stream types where sliced VBI data insertion is not supported by the device.

The following subsections specify the format of the embedded sliced VBI data.

MPEG Stream Embedded, Sliced VBI Data Format: NONE

The *V4L2_MPEG_STREAM_VBI_FMT_NONE* embedded sliced VBI format shall be interpreted by drivers as a control to cease embedding sliced VBI data in MPEG streams. Neither the device nor driver shall insert "empty" embedded sliced VBI data packets in the MPEG stream when this format is set. No MPEG stream data structures are specified for this format.

MPEG Stream Embedded, Sliced VBI Data Format: IVTV

The *V4L2_MPEG_STREAM_VBI_FMT_IVTV* embedded sliced VBI format, when supported, indicates to the driver to embed up to 36 lines of sliced VBI data per frame in an MPEG-2 *Private Stream 1 PES* packet encapsulated in an MPEG-2 *Program Pack* in the MPEG stream.

Historical context: This format specification originates from a custom, embedded, sliced VBI data format used by the `ivtv` driver. This format has already been informally specified in the kernel sources in the file `Documentation/video4linux/cx2341x/README.vbi` . The maximum size of the payload and other aspects of this format are driven by the CX23415 MPEG decoder's capabilities and limitations with respect to extracting, decoding, and displaying sliced VBI data embedded within an MPEG stream.

This format's use is *not* exclusive to the `ivtv` driver *nor* exclusive to CX2341x devices, as the sliced VBI data packet insertion into the MPEG stream is implemented in driver software. At least the `cx18` driver provides sliced VBI data insertion into an MPEG-2 PS in this format as well.

The following definitions specify the payload of the MPEG-2 *Private Stream 1 PES* packets that contain sliced VBI data when *V4L2_MPEG_STREAM_VBI_FMT_IVTV* is set. (The MPEG-2 *Private Stream 1 PES* packet header and encapsulating MPEG-2 *Program Pack* header are not detailed here. Please refer to the MPEG-2 specifications for details on those packet headers.)

The payload of the MPEG-2 *Private Stream 1 PES* packets that contain sliced VBI data is specified by struct *v4l2_mpeg_vbi_fmt_ivtv*. The payload is variable length, depending on the actual number of lines of sliced VBI data present in a video frame. The payload may be padded at the end with unspecified fill bytes to align the end of the payload to a 4-byte boundary. The payload shall never exceed 1552 bytes (2 fields with 18 lines/field with 43 bytes of data/line and a 4 byte magic number).

v4l2_mpeg_vbi_fmt_ivtv

struct v4l2_mpeg_vbi_fmt_ivtv

__u8	`magic[4]`		A "magic" constant from *Magic Constants for struct v4l2_mpeg_vbi_fmt_ivtv magic field* that indicates this is a valid sliced VBI data payload and also indicates which member of the anonymous union, `itv0` or `ITV0`, to use for the payload data.
union	(anonymous)		
	struct *v4l2_mpeg_vbi_itv0*	`itv0`	The primary form of the sliced VBI data payload that contains anywhere from 1 to 35 lines of sliced VBI data. Line masks are provided in this form of the payload indicating which VBI lines are provided.
	struct *v4l2_mpeg_vbi_ITV0*	`ITV0`	An alternate form of the sliced VBI data payload used when 36 lines of sliced VBI data are present. No line masks are provided in this form of the payload; all valid line mask bits are implcitly set.

Magic Constants for struct v4l2_mpeg_vbi_fmt_ivtv magic field

Defined Symbol	Value	Description
V4L2_MPEG_VBI_IVTV_MAGIC0	“itv0”	Indicates the itv0 member of the union in struct *v4l2_mpeg_vbi_fmt_ivtv* is valid.
V4L2_MPEG_VBI_IVTV_MAGIC1	“ITV0”	Indicates the ITV0 member of the union in struct *v4l2_mpeg_vbi_fmt_ivtv* is valid and that 36 lines of sliced VBI data are present.

v4l2_mpeg_vbi_itv0

v4l2_mpeg_vbi_ITV0

structs v4l2_mpeg_vbi_itv0 and v4l2_mpeg_vbi_ITV0

__le32	linemask[2]	Bitmasks indicating the VBI service lines present. These linemask values are stored in little endian byte order in the MPEG stream. Some reference linemask bit positions with their corresponding VBI line number and video field are given below. b_0 indicates the least significant bit of a linemask value: linemask[0] b0: line 6 first field linemask[0] b17: line 23 first field linemask[0] b18: line 6 second field linemask[0] b31: line 19 second field linemask[1] b0: line 20 second field linemask[1] b3: line 23 second field linemask[1] b4-b31: unused and set to 0
struct *v4l2_mpeg_vbi_itv0_line*	line[35]	This is a variable length array that holds from 1 to 35 lines of sliced VBI data. The sliced VBI data lines present correspond to the bits set in the linemask array, starting from b_0 of linemask[0] up through b_{31} of linemask[0], and from b_0 of linemask[1] up through b_3 of linemask[1]. line[0] corresponds to the first bit found set in the linemask array, line[1] corresponds to the second bit found set in the linemask array, etc. If no linemask array bits are set, then line[0] may contain one line of unspecified data that should be ignored by applications.

struct v4l2_mpeg_vbi_ITV0

struct *v4l2_mpeg_vbi_itv0_line*	line[36]	A fixed length array of 36 lines of sliced VBI data. line[0] through line[17] correspond to lines 6 through 23 of the first field. line[18] through line[35] corresponds to lines 6 through 23 of the second field.

v4l2_mpeg_vbi_itv0_line

struct v4l2_mpeg_vbi_itv0_line

__u8	id	A line identifier value from *Line Identifiers for struct v4l2_mpeg_vbi_itv0_line id field* that indicates the type of sliced VBI data stored on this line.
__u8	data[42]	The sliced VBI data for the line.

Line Identifiers for struct v4l2_mpeg_vbi_itv0_line id field

Defined Symbol	Value	Description
V4L2_MPEG_VBI_IVTV_TELETEXT_B	1	Refer to *Sliced VBI services* for a description of the line payload.
V4L2_MPEG_VBI_IVTV_CAPTION_525	4	Refer to *Sliced VBI services* for a description of the line payload.
V4L2_MPEG_VBI_IVTV_WSS_625	5	Refer to *Sliced VBI services* for a description of the line payload.
V4L2_MPEG_VBI_IVTV_VPS	7	Refer to *Sliced VBI services* for a description of the line payload.

Teletext Interface

This interface was aimed at devices receiving and demodulating Teletext data [*ETS 300 706* , *ITU BT.653*], evaluating the Teletext packages and storing formatted pages in cache memory. Such devices are usually implemented as microcontrollers with serial interface (I:sup:*2*C) and could be found on old TV cards, dedicated Teletext decoding cards and home-brew devices connected to the PC parallel port.

The Teletext API was designed by Martin Buck. It was defined in the kernel header file `linux/videotext.h`, the specification is available from ftp://ftp.gwdg.de/pub/linux/misc/videotext/. (Videotext is the name of the German public television Teletext service.)

Eventually the Teletext API was integrated into the V4L API with character device file names `/dev/vtx0` to `/dev/vtx31`, device major number 81, minor numbers 192 to 223.

However, teletext decoders were quickly replaced by more generic VBI demodulators and those dedicated teletext decoders no longer exist. For many years the vtx devices were still around, even though nobody used them. So the decision was made to finally remove support for the Teletext API in kernel 2.6.37.

Modern devices all use the *raw* or *Sliced VBI Data Interface* VBI API.

Radio Interface

This interface is intended for AM and FM (analog) radio receivers and transmitters.

Conventionally V4L2 radio devices are accessed through character device special files named `/dev/radio` and `/dev/radio0` to `/dev/radio63` with major number 81 and minor numbers 64 to 127.

Querying Capabilities

Devices supporting the radio interface set the `V4L2_CAP_RADIO` and `V4L2_CAP_TUNER` or `V4L2_CAP_MODULATOR` flag in the `capabilities` field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. Other combinations of capability flags are reserved for future extensions.

Supplemental Functions

Radio devices can support *controls* , and must support the *tuner or modulator* ioctls.

They do not support the video input or output, audio input or output, video standard, cropping and scaling, compression and streaming parameter, or overlay ioctls. All other ioctls and I/O methods are reserved for future extensions.

Programming

Radio devices may have a couple audio controls (as discussed in *User Controls*) such as a volume control, possibly custom controls. Further all radio devices have one tuner or modulator (these are discussed in *Tuners and Modulators*) with index number zero to select the radio frequency and to determine if a monaural or FM stereo program is received/emitted. Drivers switch automatically between AM and FM depending on the selected frequency. The *VIDIOC_G_TUNER* or *VIDIOC_G_MODULATOR* ioctl reports the supported frequency range.

RDS Interface

The Radio Data System transmits supplementary information in binary format, for example the station name or travel information, on an inaudible audio subcarrier of a radio program. This interface is aimed at devices capable of receiving and/or transmitting RDS information.

For more information see the core RDS standard *IEC 62106* and the RBDS standard *NRSC-4-B* .

Note:

Note that the RBDS standard as is used in the USA is almost identical to the RDS standard. Any RDS decoder/encoder can also handle RBDS. Only some of the fields have slightly different meanings. See the RBDS standard for more information.

The RBDS standard also specifies support for MMBS (Modified Mobile Search). This is a proprietary format which seems to be discontinued. The RDS interface does not support this format. Should support for MMBS (or the so-called 'E blocks' in general) be needed, then please contact the linux-media mailing list: https://linuxtv.org/lists.php.

Querying Capabilities

Devices supporting the RDS capturing API set the `V4L2_CAP_RDS_CAPTURE` flag in the `capabilities` field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. Any tuner that supports RDS will set the `V4L2_TUNER_CAP_RDS` flag in the `capability` field of struct *v4l2_tuner*. If the driver only passes RDS blocks without interpreting the data the `V4L2_TUNER_CAP_RDS_BLOCK_IO` flag has to be set, see *Reading RDS data* . For future use the flag `V4L2_TUNER_CAP_RDS_CONTROLS` has also been defined. However, a driver for a radio tuner with this capability does not yet exist, so if you are planning to write such a driver you should discuss this on the linux-media mailing list: https://linuxtv.org/lists.php.

Whether an RDS signal is present can be detected by looking at the `rxsubchans` field of struct *v4l2_tuner*: the `V4L2_TUNER_SUB_RDS` will be set if RDS data was detected.

Devices supporting the RDS output API set the `V4L2_CAP_RDS_OUTPUT` flag in the `capabilities` field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. Any modulator that supports RDS will set the `V4L2_TUNER_CAP_RDS` flag in the `capability` field of struct *v4l2_modulator*. In order to enable the RDS transmission one must set the `V4L2_TUNER_SUB_RDS` bit in the `txsubchans` field of struct *v4l2_modulator*. If the driver only passes RDS blocks without interpreting the data the `V4L2_TUNER_CAP_RDS_BLOCK_IO` flag has to be set. If the tuner is capable of handling RDS entities like

program identification codes and radio text, the flag V4L2_TUNER_CAP_RDS_CONTROLS should be set, see *Writing RDS data* and *FM Transmitter Control Reference* .

Reading RDS data

RDS data can be read from the radio device with the *read()* function. The data is packed in groups of three bytes.

Writing RDS data

RDS data can be written to the radio device with the *write()* function. The data is packed in groups of three bytes, as follows:

RDS datastructures

v4l2_rds_data

Table 1.33: struct v4l2_rds_data

__u8	lsb	Least Significant Byte of RDS Block
__u8	msb	Most Significant Byte of RDS Block
__u8	block	Block description

Table 1.34: Block description

Bits 0-2	Block (aka offset) of the received data.
Bits 3-5	Deprecated. Currently identical to bits 0-2. Do not use these bits.
Bit 6	Corrected bit. Indicates that an error was corrected for this data block.
Bit 7	Error bit. Indicates that an uncorrectable error occurred during reception of this block.

Table 1.35: Block defines

V4L2_RDS_BLOCK_MSK		7	Mask for bits 0-2 to get the block ID.
V4L2_RDS_BLOCK_A		0	Block A.
V4L2_RDS_BLOCK_B		1	Block B.
V4L2_RDS_BLOCK_C		2	Block C.
V4L2_RDS_BLOCK_D		3	Block D.
V4L2_RDS_BLOCK_C_ALT		4	Block C'.
V4L2_RDS_BLOCK_INVALID	read-only	7	An invalid block.
V4L2_RDS_BLOCK_CORRECTED	read-only	0x40	A bit error was detected but corrected.
V4L2_RDS_BLOCK_ERROR	read-only	0x80	An uncorrectable error occurred.

Software Defined Radio Interface (SDR)

SDR is an abbreviation of Software Defined Radio, the radio device which uses application software for modulation or demodulation. This interface is intended for controlling and data streaming of such devices.

SDR devices are accessed through character device special files named /dev/swradio0 to /dev/swradio255 with major number 81 and dynamically allocated minor numbers 0 to 255.

Querying Capabilities

Devices supporting the SDR receiver interface set the V4L2_CAP_SDR_CAPTURE and V4L2_CAP_TUNER flag in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. That

flag means the device has an Analog to Digital Converter (ADC), which is a mandatory element for the SDR receiver.

Devices supporting the SDR transmitter interface set the V4L2_CAP_SDR_OUTPUT and V4L2_CAP_MODULATOR flag in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. That flag means the device has an Digital to Analog Converter (DAC), which is a mandatory element for the SDR transmitter.

At least one of the read/write, streaming or asynchronous I/O methods must be supported.

Supplemental Functions

SDR devices can support *controls* , and must support the *Tuners and Modulators* ioctls. Tuner ioctls are used for setting the ADC/DAC sampling rate (sampling frequency) and the possible radio frequency (RF).

The V4L2_TUNER_SDR tuner type is used for setting SDR device ADC/DAC frequency, and the V4L2_TUNER_RF tuner type is used for setting radio frequency. The tuner index of the RF tuner (if any) must always follow the SDR tuner index. Normally the SDR tuner is #0 and the RF tuner is #1.

The *ioctl VIDIOC_S_HW_FREQ_SEEK* ioctl is not supported.

Data Format Negotiation

The SDR device uses the *Data Formats* ioctls to select the capture and output format. Both the sampling resolution and the data streaming format are bound to that selectable format. In addition to the basic *Data Formats* ioctls, the *ioctl VIDIOC_ENUM_FMT* ioctl must be supported as well.

To use the *Data Formats* ioctls applications set the type field of a struct *v4l2_format* to V4L2_BUF_TYPE_SDR_CAPTURE or V4L2_BUF_TYPE_SDR_OUTPUT and use the struct *v4l2_sdr_format* sdr member of the fmt union as needed per the desired operation. Currently there is two fields, pixelformat and buffersize, of struct struct *v4l2_sdr_format* which are used. Content of the pixelformat is V4L2 FourCC code of the data format. The buffersize field is maximum buffer size in bytes required for data transfer, set by the driver in order to inform application.

v4l2_sdr_format

Table 1.36: struct v4l2_sdr_format

__u32	pixelformat	The data format or type of compression, set by the application. This is a little endian *four character code* . V4L2 defines SDR formats in *SDR Formats* .
__u32	buffersize	Maximum size in bytes required for data. Value is set by the driver.
__u8	reserved[24]	This array is reserved for future extensions. Drivers and applications must set it to zero.

An SDR device may support *read/write* and/or streaming (*memory mapping* or *user pointer*) I/O.

Touch Devices

Touch devices are accessed through character device special files named /dev/v4l-touch0 to /dev/v4l-touch255 with major number 81 and dynamically allocated minor numbers 0 to 255.

Overview

Sensors may be Optical, or Projected Capacitive touch (PCT).

Processing is required to analyse the raw data and produce input events. In some systems, this may be performed on the ASIC and the raw data is purely a side-channel for diagnostics or tuning. In other systems, the ASIC is a simple analogue front end device which delivers touch data at high rate, and any touch processing must be done on the host.

For capacitive touch sensing, the touchscreen is composed of an array of horizontal and vertical conductors (alternatively called rows/columns, X/Y lines, or tx/rx). Mutual Capacitance measured is at the nodes where the conductors cross. Alternatively, Self Capacitance measures the signal from each column and row independently.

A touch input may be determined by comparing the raw capacitance measurement to a no-touch reference (or "baseline") measurement:

Delta = Raw - Reference

The reference measurement takes account of variations in the capacitance across the touch sensor matrix, for example manufacturing irregularities, environmental or edge effects.

Querying Capabilities

Devices supporting the touch interface set the `V4L2_CAP_VIDEO_CAPTURE` flag and the `V4L2_CAP_TOUCH` flag in the `capabilities` field of *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl.

At least one of the read/write or streaming I/O methods must be supported.

The formats supported by touch devices are documented in *Touch Formats* .

Data Format Negotiation

A touch device may support any I/O method.

Event Interface

The V4L2 event interface provides a means for a user to get immediately notified on certain conditions taking place on a device. This might include start of frame or loss of signal events, for example. Changes in the value or state of a V4L2 control can also be reported through events.

To receive events, the events the user is interested in first must be subscribed using the *ioctl VIDIOC_SUBSCRIBE_EVENT, VIDIOC_UNSUBSCRIBE_EVENT* ioctl. Once an event is subscribed, the events of subscribed types are dequeueable using the *ioctl VIDIOC_DQEVENT* ioctl. Events may be unsubscribed using VIDIOC_UNSUBSCRIBE_EVENT ioctl. The special event type V4L2_EVENT_ALL may be used to unsubscribe all the events the driver supports.

The event subscriptions and event queues are specific to file handles. Subscribing an event on one file handle does not affect other file handles.

The information on dequeueable events is obtained by using select or poll system calls on video devices. The V4L2 events use POLLPRI events on poll system call and exceptions on select system call.

Starting with kernel 3.1 certain guarantees can be given with regards to events:

1. Each subscribed event has its own internal dedicated event queue. This means that flooding of one event type will not interfere with other event types.
2. If the internal event queue for a particular subscribed event becomes full, then the oldest event in that queue will be dropped.
3. Where applicable, certain event types can ensure that the payload of the oldest event that is about to be dropped will be merged with the payload of the next oldest event. Thus ensuring that no information is lost, but only an intermediate step leading up to that information. See the documentation for the event you want to subscribe to whether this is applicable for that event or not.

Sub-device Interface

The complex nature of V4L2 devices, where hardware is often made of several integrated circuits that need to interact with each other in a controlled way, leads to complex V4L2 drivers. The drivers usually reflect the hardware model in software, and model the different hardware components as software blocks called sub-devices.

V4L2 sub-devices are usually kernel-only objects. If the V4L2 driver implements the media device API, they will automatically inherit from media entities. Applications will be able to enumerate the sub-devices and discover the hardware topology using the media entities, pads and links enumeration API.

In addition to make sub-devices discoverable, drivers can also choose to make them directly configurable by applications. When both the sub-device driver and the V4L2 device driver support this, sub-devices will feature a character device node on which ioctls can be called to

- query, read and write sub-devices controls
- subscribe and unsubscribe to events and retrieve them
- negotiate image formats on individual pads

Sub-device character device nodes, conventionally named `/dev/v4l-subdev*`, use major number 81.

Controls

Most V4L2 controls are implemented by sub-device hardware. Drivers usually merge all controls and expose them through video device nodes. Applications can control all sub-devices through a single interface.

Complex devices sometimes implement the same control in different pieces of hardware. This situation is common in embedded platforms, where both sensors and image processing hardware implement identical functions, such as contrast adjustment, white balance or faulty pixels correction. As the V4L2 controls API doesn't support several identical controls in a single device, all but one of the identical controls are hidden.

Applications can access those hidden controls through the sub-device node with the V4L2 control API described in *User Controls* . The ioctls behave identically as when issued on V4L2 device nodes, with the exception that they deal only with controls implemented in the sub-device.

Depending on the driver, those controls might also be exposed through one (or several) V4L2 device nodes.

Events

V4L2 sub-devices can notify applications of events as described in *Event Interface* . The API behaves identically as when used on V4L2 device nodes, with the exception that it only deals with events generated by the sub-device. Depending on the driver, those events might also be reported on one (or several) V4L2 device nodes.

Pad-level Formats

Warning:

Pad-level formats are only applicable to very complex devices that need to expose low-level format configuration to user space. Generic V4L2 applications do not *need to use the API described in this section.*

Note:

For the purpose of this section, the term format *means the combination of media bus data format, frame width and frame height.*

Image formats are typically negotiated on video capture and output devices using the format and *selection* ioctls. The driver is responsible for configuring every block in the video pipeline according to the requested format at the pipeline input and/or output.

For complex devices, such as often found in embedded systems, identical image sizes at the output of a pipeline can be achieved using different hardware configurations. One such example is shown on *Image Format Negotiation on Pipelines* , where image scaling can be performed on both the video sensor and the host image processing hardware.

Fig. 1.9: Image Format Negotiation on Pipelines
High quality and high speed pipeline configuration

The sensor scaler is usually of less quality than the host scaler, but scaling on the sensor is required to achieve higher frame rates. Depending on the use case (quality vs. speed), the pipeline must be configured differently. Applications need to configure the formats at every point in the pipeline explicitly.

Drivers that implement the *media API* can expose pad-level image format configuration to applications. When they do, applications can use the *VIDIOC_SUBDEV_G_FMT* and *VIDIOC_SUBDEV_S_FMT* ioctls. to negotiate formats on a per-pad basis.

Applications are responsible for configuring coherent parameters on the whole pipeline and making sure that connected pads have compatible formats. The pipeline is checked for formats mismatch at *VIDIOC_STREAMON* time, and an EPIPE error code is then returned if the configuration is invalid.

Pad-level image format configuration support can be tested by calling the *ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT* ioctl on pad 0. If the driver returns an EINVAL error code pad-level format configuration is not supported by the sub-device.

Format Negotiation

Acceptable formats on pads can (and usually do) depend on a number of external parameters, such as formats on other pads, active links, or even controls. Finding a combination of formats on all pads in a video pipeline, acceptable to both application and driver, can't rely on formats enumeration only. A format negotiation mechanism is required.

Central to the format negotiation mechanism are the get/set format operations. When called with the `which` argument set to *V4L2_SUBDEV_FORMAT_TRY* , the *VIDIOC_SUBDEV_G_FMT* and *VIDIOC_SUBDEV_S_FMT* ioctls operate on a set of formats parameters that are not connected to the hardware configuration. Modifying those 'try' formats leaves the device state untouched (this applies to both the software state stored in the driver and the hardware state stored in the device itself).

While not kept as part of the device state, try formats are stored in the sub-device file handles. A *VIDIOC_SUBDEV_G_FMT* call will return the last try format set *on the same sub-device file handle*. Several applications querying the same sub-device at the same time will thus not interact with each other.

To find out whether a particular format is supported by the device, applications use the *VIDIOC_SUBDEV_S_FMT* ioctl. Drivers verify and, if needed, change the requested `format` based on device requirements and return the possibly modified value. Applications can then choose to try a different format or accept the returned value and continue.

Formats returned by the driver during a negotiation iteration are guaranteed to be supported by the device. In particular, drivers guarantee that a returned format will not be further changed if passed to

an *VIDIOC_SUBDEV_S_FMT* call as-is (as long as external parameters, such as formats on other pads or links' configuration are not changed).

Drivers automatically propagate formats inside sub-devices. When a try or active format is set on a pad, corresponding formats on other pads of the same sub-device can be modified by the driver. Drivers are free to modify formats as required by the device. However, they should comply with the following rules when possible:

- Formats should be propagated from sink pads to source pads. Modifying a format on a source pad should not modify the format on any sink pad.
- Sub-devices that scale frames using variable scaling factors should reset the scale factors to default values when sink pads formats are modified. If the 1:1 scaling ratio is supported, this means that source pads formats should be reset to the sink pads formats.

Formats are not propagated across links, as that would involve propagating them from one sub-device file handle to another. Applications must then take care to configure both ends of every link explicitly with compatible formats. Identical formats on the two ends of a link are guaranteed to be compatible. Drivers are free to accept different formats matching device requirements as being compatible.

Sample Pipeline Configuration shows a sample configuration sequence for the pipeline described in *Image Format Negotiation on Pipelines* (table columns list entity names and pad numbers).

Table 1.37: Sample Pipeline Configuration

	Sensor/0 format	Frontend/0 format	Frontend/1 format	Scaler/0 format	Scaler/0 compose selection rectangle	Scaler/1 format
Initial state	2048x1536/SGRBG8_1X8	(default)	(default)	(default)	(default)	(default)
Configure frontend sink format	2048x1536/SGRBG8_1X8	*2048x1536/SGRBG8_1X8*	*2046x1534/SGRBG8_1X8*	(default)	(default)	(default)
Configure scaler sink format	2048x1536/SGRBG8_1X8	2048x1536/SGRBG8_1X8	2046x1534/SGRBG8_1X8	*2046x1534/SGRBG8_1X8*	*0,0/2046x1534*	*2046x1534/SGRBG8_1X8*
Configure scaler sink compose selection	2048x1536/SGRBG8_1X8	2048x1536/SGRBG8_1X8	2046x1534/SGRBG8_1X8	2046x1534/SGRBG8_1X8	*0,0/1280x960*	*1280x960/SGRBG8_1X8*

1. Initial state. The sensor source pad format is set to its native 3MP size and V4L2_MBUS_FMT_SGRBG8_1X8 media bus code. Formats on the host frontend and scaler sink and source pads have the default values, as well as the compose rectangle on the scaler's sink pad.
2. The application configures the frontend sink pad format's size to 2048x1536 and its media bus code to V4L2_MBUS_FMT_SGRBG_1X8. The driver propagates the format to the frontend source pad.
3. The application configures the scaler sink pad format's size to 2046x1534 and the media bus code to V4L2_MBUS_FMT_SGRBG_1X8 to match the frontend source size and media bus code. The media bus code on the sink pad is set to V4L2_MBUS_FMT_SGRBG_1X8. The driver propagates the size to the compose selection rectangle on the scaler's sink pad, and the format to the scaler source pad.
4. The application configures the size of the compose selection rectangle of the scaler's sink pad 1280x960. The driver propagates the size to the scaler's source pad format.

When satisfied with the try results, applications can set the active formats by setting the `which` argument to `V4L2_SUBDEV_FORMAT_ACTIVE`. Active formats are changed exactly as try formats by drivers. To avoid modifying the hardware state during format negotiation, applications should negotiate try formats first and then modify the active settings using the try formats returned during the last negotiation iteration. This guarantees that the active format will be applied as-is by the driver without being modified.

Selections: cropping, scaling and composition

Many sub-devices support cropping frames on their input or output pads (or possible even on both). Cropping is used to select the area of interest in an image, typically on an image sensor or a video decoder. It can also be used as part of digital zoom implementations to select the area of the image that will be scaled up.

Crop settings are defined by a crop rectangle and represented in a struct *v4l2_rect* by the coordinates of the top left corner and the rectangle size. Both the coordinates and sizes are expressed in pixels.

As for pad formats, drivers store try and active rectangles for the selection targets *Common selection definitions* .

On sink pads, cropping is applied relative to the current pad format. The pad format represents the image size as received by the sub-device from the previous block in the pipeline, and the crop rectangle represents the sub-image that will be transmitted further inside the sub-device for processing.

The scaling operation changes the size of the image by scaling it to new dimensions. The scaling ratio isn't specified explicitly, but is implied from the original and scaled image sizes. Both sizes are represented by struct *v4l2_rect*.

Scaling support is optional. When supported by a subdev, the crop rectangle on the subdev's sink pad is scaled to the size configured using the *VIDIOC_SUBDEV_S_SELECTION* IOCTL using V4L2_SEL_TGT_COMPOSE selection target on the same pad. If the subdev supports scaling but not composing, the top and left values are not used and must always be set to zero.

On source pads, cropping is similar to sink pads, with the exception that the source size from which the cropping is performed, is the COMPOSE rectangle on the sink pad. In both sink and source pads, the crop rectangle must be entirely contained inside the source image size for the crop operation.

The drivers should always use the closest possible rectangle the user requests on all selection targets, unless specifically told otherwise. V4L2_SEL_FLAG_GE and V4L2_SEL_FLAG_LE flags may be used to round the image size either up or down. *Selection flags*

Types of selection targets

Actual targets

Actual targets (without a postfix) reflect the actual hardware configuration at any point of time. There is a BOUNDS target corresponding to every actual target.

BOUNDS targets

BOUNDS targets is the smallest rectangle that contains all valid actual rectangles. It may not be possible to set the actual rectangle as large as the BOUNDS rectangle, however. This may be because e.g. a sensor's pixel array is not rectangular but cross-shaped or round. The maximum size may also be smaller than the BOUNDS rectangle.

Order of configuration and format propagation

Inside subdevs, the order of image processing steps will always be from the sink pad towards the source pad. This is also reflected in the order in which the configuration must be performed by the user: the changes made will be propagated to any subsequent stages. If this behaviour is not desired, the user must set V4L2_SEL_FLAG_KEEP_CONFIG flag. This flag causes no propagation of the changes are allowed in any circumstances. This may also cause the accessed rectangle to be adjusted by the driver, depending on the properties of the underlying hardware.

The coordinates to a step always refer to the actual size of the previous step. The exception to this rule is the source compose rectangle, which refers to the sink compose bounds rectangle — if it is supported by the hardware.

1. Sink pad format. The user configures the sink pad format. This format defines the parameters of the image the entity receives through the pad for further processing.
2. Sink pad actual crop selection. The sink pad crop defines the crop performed to the sink pad format.
3. Sink pad actual compose selection. The size of the sink pad compose rectangle defines the scaling ratio compared to the size of the sink pad crop rectangle. The location of the compose rectangle specifies the location of the actual sink compose rectangle in the sink compose bounds rectangle.

4. Source pad actual crop selection. Crop on the source pad defines crop performed to the image in the sink compose bounds rectangle.
5. Source pad format. The source pad format defines the output pixel format of the subdev, as well as the other parameters with the exception of the image width and height. Width and height are defined by the size of the source pad actual crop selection.

Accessing any of the above rectangles not supported by the subdev will return EINVAL. Any rectangle referring to a previous unsupported rectangle coordinates will instead refer to the previous supported rectangle. For example, if sink crop is not supported, the compose selection will refer to the sink pad format dimensions instead.

Fig. 1.10: **Figure 4.5. Image processing in subdevs: simple crop example**

In the above example, the subdev supports cropping on its sink pad. To configure it, the user sets the media bus format on the subdev's sink pad. Now the actual crop rectangle can be set on the sink pad — the location and size of this rectangle reflect the location and size of a rectangle to be cropped from the sink format. The size of the sink crop rectangle will also be the size of the format of the subdev's source pad.

Fig. 1.11: **Figure 4.6. Image processing in subdevs: scaling with multiple sources**

In this example, the subdev is capable of first cropping, then scaling and finally cropping for two source pads individually from the resulting scaled image. The location of the scaled image in the cropped image is ignored in sink compose target. Both of the locations of the source crop rectangles refer to the sink scaling rectangle, independently cropping an area at location specified by the source crop rectangle from it.

Fig. 1.12: **Figure 4.7. Image processing in subdevs: scaling and composition with multiple sinks and sources**

The subdev driver supports two sink pads and two source pads. The images from both of the sink pads are individually cropped, then scaled and further composed on the composition bounds rectangle. From that, two independent streams are cropped and sent out of the subdev from the source pads.

Media Bus Formats

v4l2_mbus_framefmt

Table 1.38: struct v4l2_mbus_framefmt

__u32	width	Image width, in pixels.
__u32	height	Image height, in pixels.
__u32	code	Format code, from enum *v4l2_mbus_pixelcode* .
__u32	field	Field order, from enum *v4l2_field*. See *Field Order* for details.
__u32	colorspace	Image colorspace, from enum *v4l2_colorspace*. See *Colorspaces* for details.
enum *v4l2_ycbcr_encoding*	ycbcr_enc	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
enum *v4l2_quantization*	quantization	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
enum *v4l2_xfer_func*	xfer_func	This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces* .
__u16	reserved[11]	Reserved for future extensions. Applications and drivers must set the array to zero.

Media Bus Pixel Codes

The media bus pixel codes describe image formats as flowing over physical busses (both between separate physical components and inside SoC devices). This should not be confused with the V4L2 pixel formats that describe, using four character codes, image formats as stored in memory.

While there is a relationship between image formats on busses and image formats in memory (a raw Bayer image won't be magically converted to JPEG just by storing it to memory), there is no one-to-one correspondance between them.

Packed RGB Formats

Those formats transfer pixel data as red, green and blue components. The format code is made of the following information.

- The red, green and blue components order code, as encoded in a pixel sample. Possible values are RGB and BGR.
- The number of bits per component, for each component. The values can be different for all components. Common values are 555 and 565.
- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. Common values are 1 and 2.
- The bus width.
- For formats where the total number of bits per pixel is smaller than the number of bus samples per pixel times the bus width, a padding value stating if the bytes are padded in their most high order bits (PADHI) or low order bits (PADLO). A "C" prefix is used for component-wise padding in the most high order bits (CPADHI) or low order bits (CPADLO) of each separate component.
- For formats where the number of bus samples per pixel is larger than 1, an endianness value stating if the pixel is transferred MSB first (BE) or LSB first (LE).

For instance, a format where pixels are encoded as 5-bits red, 5-bits green and 5-bit blue values padded on the high bit, transferred as 2 8-bit samples per pixel with the most significant bits (padding, red and half of the green value) transferred first will be named `MEDIA_BUS_FMT_RGB555_2X8_PADHI_BE`.

The following tables list existing packed RGB formats.

Table 1.39: RGB formats

Identifier	Code	Bit	Data organization																													
			31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDIA_BUS_FMT_RGB444_1X12	0x1016																						r_3	r_2	r_1	r_0	g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB444_2X8_PADHI_BE	0x1001																										0	0	0	0				
																															r_3	r_2	r_1	r_0
																											g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB444_2X8_PADHI_LE	0x1002																										g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
																											0	0	0	0	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_RGB555_2X8_PADHI_BE	0x1003																										0							
																												r_4	r_3	r_2	r_1	r_0	g_4	g_3
																											g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB555_2X8_PADHI_LE	0x1004																										g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
																											0	r_4	r_3	r_2	r_1	r_0	g_4	g_3
MEDIA_BUS_FMT_RGB565_1X16	0x1017																		r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_BGR565_2X8_BE	0x1005																										b_4	b_3	b_2	b_1	b_0	g_5	g_4	g_3
																											g_2	g_1	g_0	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_BGR565_2X8_LE	0x1006																										g_2	g_1	g_0	r_4	r_3	r_2	r_1	r_0
																											b_4	b_3	b_2	b_1	b_0	g_5	g_4	g_3
MEDIA_BUS_FMT_RGB565_2X8_BE	0x1007																										r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3
																											g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB565_2X8_LE	0x1008																										g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
																											r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3
MEDIA_BUS_FMT_RGB666_1X18	0x1009																r_5	r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3	g_2	g_1	g_0	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RBG888_1X24	0x100e										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_RGB666_1X24_CPADHI	0x1015										0	0	r_5	r_4	r_3	r_2	r_1	r_0	0	0	g_5	g_4	g_3	g_2	g_1	g_0	0	0	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_BGR888_1X24	0x1013										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_GBR888_1X24	0x1014										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_RGB888_1X24	0x100a										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB888_2X12_BE	0x100b																						r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4
																							g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB888_2X12_LE	0x100c																						g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
																							r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4
MEDIA_BUS_FMT_ARGB888_1X32	0x100d		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB888_1X32_PADHI	0x100f		0	0	0	0	0	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

On LVDS buses, usually each sample is transferred serialized in seven time slots per pixel clock, on three (18-bit) or four (24-bit) differential data pairs at the same time. The remaining bits are used for control signals as defined by SPWG/PSWG/VESA or JEIDA standards. The 24-bit RGB format serialized in seven time slots on four lanes using JEIDA defined bit mapping will be named `MEDIA_BUS_FMT_RGB888_1X7X4_JEIDA`, for example.

Table 1.40: LVDS RGB formats

Identifier	Code			Data organization			
		Timeslot	Lane	3	2	1	0
MEDIA_BUS_FMT_RGB666_1X7X3_SPWG	0x1010	0			d	b_1	g_0
		1			d	b_0	r_5
		2			d	g_5	r_4
		3			b_5	g_4	r_3
		4			b_4	g_3	r_2
		5			b_3	g_2	r_1
		6			b_2	g_1	r_0
MEDIA_BUS_FMT_RGB888_1X7X4_SPWG	0x1011	0		d	d	b_1	g_0
		1		b_7	d	b_0	r_5
		2		b_6	d	g_5	r_4
		3		g_7	b_5	g_4	r_3
		4		g_6	b_4	g_3	r_2
		5		r_7	b_3	g_2	r_1
		6		r_6	b_2	g_1	r_0
MEDIA_BUS_FMT_RGB888_1X7X4_JEIDA	0x1012	0		d	d	b_3	g_2
		1		b_1	d	b_2	r_7
		2		b_0	d	g_7	r_6
		3		g_1	b_7	g_6	r_5
		4		g_0	b_6	g_5	r_4
		5		r_1	b_5	g_4	r_3
		6		r_0	b_4	g_3	r_2

Bayer Formats

Those formats transfer pixel data as red, green and blue components. The format code is made of the following information.

- The red, green and blue components order code, as encoded in a pixel sample. The possible values are shown in *Figure 4.8 Bayer Patterns* .
- The number of bits per pixel component. All components are transferred on the same number of bits. Common values are 8, 10 and 12.
- The compression (optional). If the pixel components are ALAW- or DPCM-compressed, a mention of the compression scheme and the number of bits per compressed pixel component.
- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. Common values are 1 and 2.
- The bus width.
- For formats where the total number of bits per pixel is smaller than the number of bus samples per pixel times the bus width, a padding value stating if the bytes are padded in their most high order bits (PADHI) or low order bits (PADLO).
- For formats where the number of bus samples per pixel is larger than 1, an endianness value stating if the pixel is transferred MSB first (BE) or LSB first (LE).

For instance, a format with uncompressed 10-bit Bayer components arranged in a red, green, green, blue pattern transferred as 2 8-bit samples per pixel with the least significant bits transferred first will be named `MEDIA_BUS_FMT_SRGGB10_2X8_PADHI_LE`.

Fig. 1.13: **Figure 4.8 Bayer Patterns**

The following table lists existing packed Bayer formats. The data organization is given as an example for the first pixel only.

Table 1.41: Bayer Formats

Identifier	Code		Data organization															
		Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDIA_BUS_FMT_SBGGR8_1X8	0x3001										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG8_1X8	0x3013										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG8_1X8	0x3002										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB8_1X8	0x3014										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR10_ALAW8_1X8	0x3015										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG10_ALAW8_1X8	0x3016										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG10_ALAW8_1X8	0x3017										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB10_ALAW8_1X8	0x3018										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR10_DPCM8_1X8	0x300b										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG10_DPCM8_1X8	0x300c										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG10_DPCM8_1X8	0x3009										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB10_DPCM8_1X8	0x300d										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR10_2X8_PADHI_BE	0x3003										0	0	0	0	0	0	b_9	b_8
											b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SBGGR10_2X8_PADHI_LE	0x3004										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
											0	0	0	0	0	0	b_9	b_8
MEDIA_BUS_FMT_SBGGR10_2X8_PADLO_BE	0x3005										b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2
											b_1	b_0	0	0	0	0	0	0
MEDIA_BUS_FMT_SBGGR10_2X8_PADLO_LE	0x3006										b_1	b_0	0	0	0	0	0	0
											b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2
MEDIA_BUS_FMT_SBGGR10_1X10	0x3007								b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG10_1X10	0x300e								g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0

Continued on next page

Table 1.41 – continued from previous page

Identifier	Code		Data organization															
		Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDIA_BUS_FMT_SGRBG10_1X10	0x300a								g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB10_1X10	0x300f								r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR12_1X12	0x3008						b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG12_1X12	0x3010						g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG12_1X12	0x3011						g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB12_1X12	0x3012						r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR14_1X14	0x3019				b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG14_1X14	0x301a				g_{13}	g_{12}	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG14_1X14	0x301b				g_{13}	g_{12}	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB14_1X14	0x301c				r_{13}	r_{12}	r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR16_1X16	0x301d		b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG16_1X16	0x301e		g_{15}	g_{14}	g_{13}	g_{12}	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG16_1X16	0x301f		g_{15}	g_{14}	g_{13}	g_{12}	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB16_1X16	0x3020		r_{15}	r_{14}	r_{13}	r_{12}	r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0

Packed YUV Formats

Those data formats transfer pixel data as (possibly downsampled) Y, U and V components. Some formats include dummy bits in some of their samples and are collectively referred to as “YDYC” (Y-Dummy-Y-Chroma) formats. One cannot rely on the values of these dummy bits as those are undefined.

The format code is made of the following information.

- The Y, U and V components order code, as transferred on the bus. Possible values are YUYV, UYVY, YVYU and VYUY for formats with no dummy bit, and YDYUYDYV, YDYVYDYU, YUYDYVYD and YVYDYUYD for YDYC formats.
- The number of bits per pixel component. All components are transferred on the same number of bits. Common values are 8, 10 and 12.
- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. Common values are 1, 1.5 (encoded as 1_5) and 2.
- The bus width. When the bus width is larger than the number of bits per pixel component, several components are packed in a single bus sample. The components are ordered as specified by the order code, with components on the left of the code transferred in the high order bits. Common values are 8 and 16.

For instance, a format where pixels are encoded as 8-bit YUV values downsampled to 4:2:2 and transferred as 2 8-bit bus samples per pixel in the U, Y, V, Y order will be named `MEDIA_BUS_FMT_UYVY8_2X8`.

YUV Formats lists existing packed YUV formats and describes the organization of each pixel data in each sample. When a format pattern is split across multiple samples each of the samples in the pattern is described.

The role of each bit transferred over the bus is identified by one of the following codes.

- y_x for luma component bit number x
- u_x for blue chroma component bit number x

- v_x for red chroma component bit number x
- a_x for alpha component bit number x
- for non-available bits (for positions higher than the bus width)
- d for dummy bits

Table 1.42: YUV Formats

Identifier	Code	Bit	Data organization																													
			31	30	29	28	27	26	25	24	23	22	21	10	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDIA_BUS_FMT_Y8_1X8	0x2001																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_UV8_1X8	0x2015																										u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_UYVY8_1_5X8	0x2002																										u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY8_1_5X8	0x2003																										v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV8_1_5X8	0x2004																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU8_1_5X8	0x2005																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_UYVY8_2X8	0x2006																										u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY8_2X8	0x2007																										v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV8_2X8	0x2008																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU8_2X8	0x2009																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_Y10_1X10	0x200a																								y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_UYVY10_2X10	0x2018																								u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY10_2X10	0x2019																								v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV10_2X10	0x200b																								y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU10_2X10	0x200c																								y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_Y12_1X12	0x2013																						y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_UYVY12_2X12	0x201c																						u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY12_2X12	0x201d																						v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV12_2X12	0x201e																						y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0

Continued on next page

Table 1.42 – continued from previous page

Identifier	Code	Bit	Data organization																																					
			31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																							v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU12_2X12	0x201f																						y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_UYVY8_1X16	0x200f																		u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																			v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY8_1X16	0x2010																		v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																			u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV8_1X16	0x2011																		y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU8_1X16	0x2012																		y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_YDYUYDYV8_1X16	0x2014																		y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	d	d	d	d	d	d	d	d
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	d	d	d	d	d	d	d	d
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_UYVY10_1X20	0x201a														u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
															v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY10_1X20	0x201b														v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
															u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV10_1X20	0x200d														y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
															y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU10_1X20	0x200e														y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
															y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_VUY8_1X24	0x201a										v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUV8_1X24	0x2025										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_UYVY12_1X24	0x2020										u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
											v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY12_1X24	0x2021										v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
											u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV12_1X24	0x2022										y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
											y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU12_1X24	0x2023										y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
											y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_YUV10_1X30	0x2016					y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_AYUV8_1X32	0x2017		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0

HSV/HSL Formats

Those formats transfer pixel data as RGB values in a cylindrical-coordinate system using Hue-Saturation-Value or Hue-Saturation-Lightness components. The format code is made of the following information.

- The hue, saturation, value or lightness and optional alpha components order code, as encoded in a pixel sample. The only currently supported value is AHSV.
- The number of bits per component, for each component. The values can be different for all components. The only currently supported value is 8888.
- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. The only currently supported value is 1.
- The bus width.
- For formats where the total number of bits per pixel is smaller than the number of bus samples per pixel times the bus width, a padding value stating if the bytes are padded in their most high order bits (PADHI) or low order bits (PADLO).
- For formats where the number of bus samples per pixel is larger than 1, an endianness value stating if the pixel is transferred MSB first (BE) or LSB first (LE).

The following table lists existing HSV/HSL formats.

Table 1.43: HSV/HSL formats

Identifier	Code	Bit	Data organization																															
			31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDIA_BUS_FMT_AHSV8888_1X32	0x6001		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	h_7	h_6	h_5	h_4	h_3	h_2	h_1	h_0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0

JPEG Compressed Formats

Those data formats consist of an ordered sequence of 8-bit bytes obtained from JPEG compression process. Additionally to the _JPEG postfix the format code is made of the following information.

- The number of bus samples per entropy encoded byte.
- The bus width.

For instance, for a JPEG baseline process and an 8-bit bus width the format will be named MEDIA_BUS_FMT_JPEG_1X8.

The following table lists existing JPEG compressed formats.

Table 1.44: JPEG Formats

Identifier	Code	Remarks
MEDIA_BUS_FMT_JPEG_1X8	0x4001	Besides of its usage for the parallel bus this format is recommended for transmission of JPEG data over MIPI CSI bus using the User Defined 8-bit Data types.

Vendor and Device Specific Formats

This section lists complex data formats that are either vendor or device specific.

The following table lists the existing vendor and device specific formats.

Table 1.45: Vendor and device specific formats

Identifier	Code	Comments
MEDIA_BUS_FMT_S5C_UYVY_JPEG_1X8	0x5001	Interleaved raw UYVY and JPEG image format with embedded meta-data used by Samsung S3C73MX camera sensors.

1.2.5 V4L2 Driver Programming

to do

1.2.6 Libv4l Userspace Library

Introduction

libv4l is a collection of libraries which adds a thin abstraction layer on top of video4linux2 devices. The purpose of this (thin) layer is to make it easy for application writers to support a wide variety of devices without having to write separate code for different devices in the same class.

An example of using libv4l is provided by *v4l2grab* .

libv4l consists of 3 different libraries:

libv4lconvert

libv4lconvert is a library that converts several different pixelformats found in V4L2 drivers into a few common RGB and YUY formats.

It currently accepts the following V4L2 driver formats: *V4L2_PIX_FMT_BGR24*, *V4L2_PIX_FMT_HM12*, *V4L2_PIX_FMT_JPEG*, *V4L2_PIX_FMT_MJPEG*, *V4L2_PIX_FMT_MR97310A*, *V4L2_PIX_FMT_OV511*, *V4L2_PIX_FMT_OV518*, *V4L2_PIX_FMT_PAC207*, *V4L2_PIX_FMT_PJPG*, *V4L2_PIX_FMT_RGB24*, *V4L2_PIX_FMT_SBGGR8*, *V4L2_PIX_FMT_SGBRG8*, *V4L2_PIX_FMT_SGRBG8*, *V4L2_PIX_FMT_SN9C10X*, *V4L2_PIX_FMT_SN9C20X_I420*, *V4L2_PIX_FMT_SPCA501*, *V4L2_PIX_FMT_SPCA505*, *V4L2_PIX_FMT_SPCA508*, *V4L2_PIX_FMT_SPCA561*, *V4L2_PIX_FMT_SQ905C*, *V4L2_PIX_FMT_SRGGB8*, *V4L2_PIX_FMT_UYVY*, *V4L2_PIX_FMT_YUV420*, *V4L2_PIX_FMT_YUYV*, *V4L2_PIX_FMT_YVU420*, and *V4L2_PIX_FMT_YVYU*.

Later on libv4lconvert was expanded to also be able to do various video processing functions to improve webcam video quality. The video processing is split in to 2 parts: libv4lconvert/control and libv4lconvert/processing.

The control part is used to offer video controls which can be used to control the video processing functions made available by libv4lconvert/processing. These controls are stored application wide (until reboot) by using a persistent shared memory object.

libv4lconvert/processing offers the actual video processing functionality.

libv4l1

This library offers functions that can be used to quickly make v4l1 applications work with v4l2 devices. These functions work exactly like the normal open/close/etc, except that libv4l1 does full emulation of the v4l1 api on top of v4l2 drivers, in case of v4l1 drivers it will just pass calls through.

Since those functions are emulations of the old V4L1 API, it shouldn't be used for new applications.

libv4l2

This library should be used for all modern V4L2 applications.

It provides handles to call V4L2 open/ioctl/close/poll methods. Instead of just providing the raw output of the device, it enhances the calls in the sense that it will use libv4lconvert to provide more video formats and to enhance the image quality.

In most cases, libv4l2 just passes the calls directly through to the v4l2 driver, intercepting the calls to *VIDIOC_TRY_FMT*, *VIDIOC_G_FMT*, *VIDIOC_S_FMT*, *VIDIOC_ENUM_FRAMESIZES* and *VIDIOC_ENUM_FRAMEINTERVALS* in order to emulate the formats *V4L2_PIX_FMT_BGR24*, *V4L2_PIX_FMT_RGB24*, *V4L2_PIX_FMT_YUV420*, and *V4L2_PIX_FMT_YVU420*, if they aren't available in the driver. *VIDIOC_ENUM_FMT* keeps enumerating the hardware supported formats, plus the emulated formats offered by libv4l at the end.

Libv4l device control functions

The common file operation methods are provided by libv4l.

Those functions operate just like the gcc function dup() and V4L2 functions *open()*, *close()*, *ioctl()*, *read()*, *mmap()* and *munmap()*:

int **v4l2_open**(const char *_file_, int _oflag_, ...)
operates like the *open()* function.

int **v4l2_close**(int _fd_)
operates like the *close()* function.

int **v4l2_dup**(int *fd*)
operates like the libc dup() function, duplicating a file handler.

int **v4l2_ioctl**(int *fd*, unsigned long int *request*, ...)
operates like the *ioctl()* function.

int **v4l2_read**(int *fd*, void* *buffer*, size_t *n*)
operates like the *read()* function.

void v4l2_mmap(void *start, size_t length, int prot, int flags, int fd, int64_t offset);
operates like the *munmap()* function.

int v4l2_munmap(void *_start, size_t length);
operates like the *munmap()* function.

Those functions provide additional control:

int **v4l2_fd_open**(int *fd*, int *v4l2_flags*)
opens an already opened fd for further use through v4l2lib and possibly modify libv4l2's default behavior through the v4l2_flags argument. Currently, v4l2_flags can be V4L2_DISABLE_CONVERSION, to disable format conversion.

int **v4l2_set_control**(int *fd*, int *cid*, int *value*)
This function takes a value of 0 - 65535, and then scales that range to the actual range of the given v4l control id, and then if the cid exists and is not locked sets the cid to the scaled value.

int **v4l2_get_control**(int *fd*, int *cid*)
This function returns a value of 0 - 65535, scaled to from the actual range of the given v4l control id. when the cid does not exist, could not be accessed for some reason, or some error occurred 0 is returned.

v4l1compat.so wrapper library

This library intercepts calls to *open()*, *close()*, *ioctl()*, *mmap()* and *munmap()* operations and redirects them to the libv4l counterparts, by using LD_PRELOAD=/usr/lib/v4l1compat.so. It also emulates V4L1 calls via V4L2 API.

It allows usage of binary legacy applications that still don't use libv4l.

1.2.7 Changes

The following chapters document the evolution of the V4L2 API, errata or extensions. They are also intended to help application and driver writers to port or update their code.

Differences between V4L and V4L2

The Video For Linux API was first introduced in Linux 2.1 to unify and replace various TV and radio device related interfaces, developed independently by driver writers in prior years. Starting with Linux 2.5 the much improved V4L2 API replaces the V4L API. The support for the old V4L calls were removed from Kernel, but the library *Libv4l Userspace Library* supports the conversion of a V4L API system call into a V4L2 one.

Opening and Closing Devices

For compatibility reasons the character device file names recommended for V4L2 video capture, overlay, radio and raw vbi capture devices did not change from those used by V4L. They are listed in *Interfaces* and below in *V4L Device Types, Names and Numbers* .

The teletext devices (minor range 192-223) have been removed in V4L2 and no longer exist. There is no hardware available anymore for handling pure teletext. Instead raw or sliced VBI is used.

The V4L videodev module automatically assigns minor numbers to drivers in load order, depending on the registered device type. We recommend that V4L2 drivers by default register devices with the same numbers, but the system administrator can assign arbitrary minor numbers using driver module options. The major device number remains 81.

Table 1.46: V4L Device Types, Names and Numbers

Device Type	File Name	Minor Numbers
Video capture and overlay	/dev/video and /dev/bttv0 [1], /dev/video0 to /dev/video63	0-63
Radio receiver	/dev/radio [2], /dev/radio0 to /dev/radio63	64-127
Raw VBI capture	/dev/vbi, /dev/vbi0 to /dev/vbi31	224-255

V4L prohibits (or used to prohibit) multiple opens of a device file. V4L2 drivers *may* support multiple opens, see *Opening and Closing Devices* for details and consequences.

V4L drivers respond to V4L2 ioctls with an EINVAL error code.

Querying Capabilities

The V4L VIDIOCGCAP ioctl is equivalent to V4L2's *ioctl VIDIOC_QUERYCAP* .

The name field in struct video_capability became card in struct *v4l2_capability*, type was replaced by capabilities. Note V4L2 does not distinguish between device types like this, better think of basic video input, video output and radio devices supporting a set of related functions like video capturing, video overlay and VBI capturing. See *Opening and Closing Devices* for an introduction.

struct video_capability type	struct *v4l2_capability* capabilities flags	Purpose
VID_TYPE_CAPTURE	V4L2_CAP_VIDEO_CAPTURE	The *video capture* interface is supported.
VID_TYPE_TUNER	V4L2_CAP_TUNER	The device has a *tuner or modulator* .
VID_TYPE_TELETEXT	V4L2_CAP_VBI_CAPTURE	The *raw VBI capture* interface is supported.
VID_TYPE_OVERLAY	V4L2_CAP_VIDEO_OVERLAY	The *video overlay* interface is supported.
VID_TYPE_CHROMAKEY	V4L2_FBUF_CAP_CHROMAKEY in field capability of struct *v4l2_framebuffer*	Whether chromakey overlay is supported. For more information on overlay see *Video Overlay Interface* .
VID_TYPE_CLIPPING	V4L2_FBUF_CAP_LIST_CLIPPING and V4L2_FBUF_CAP_BITMAP_CLIPPING in field capability of struct *v4l2_framebuffer*	Whether clipping the overlaid image is supported, see *Video Overlay Interface* .
VID_TYPE_FRAMERAM	V4L2_FBUF_CAP_EXTERNOVERLAY *not set* in field capability of struct *v4l2_framebuffer*	Whether overlay overwrites frame buffer memory, see *Video Overlay Interface* .

Continued on next page

[1] According to Documentation/admin-guide/devices.rst these should be symbolic links to /dev/video0. Note the original bttv interface is not compatible with V4L or V4L2.

[2] According to Documentation/admin-guide/devices.rst a symbolic link to /dev/radio0.

Table 1.47 – continued from previous page

struct video_capability type	struct *v4l2_capability* capabilities flags	Purpose
VID_TYPE_SCALES	-	This flag indicates if the hardware can scale images. The V4L2 API implies the scale factor by setting the cropping dimensions and image size with the *VIDIOC_S_CROP* and *VIDIOC_S_FMT* ioctl, respectively. The driver returns the closest sizes possible. For more information on cropping and scaling see *Image Cropping, Insertion and Scaling* .
VID_TYPE_MONOCHROME	-	Applications can enumerate the supported image formats with the *ioctl VIDIOC_ENUM_FMT* ioctl to determine if the device supports grey scale capturing only. For more information on image formats see *Image Formats* .
VID_TYPE_SUBCAPTURE	-	Applications can call the *VIDIOC_G_CROP* ioctl to determine if the device supports capturing a subsection of the full picture ("cropping" in V4L2). If not, the ioctl returns the EINVAL error code. For more information on cropping and scaling see *Image Cropping, Insertion and Scaling* .
VID_TYPE_MPEG_DECODER	-	Applications can enumerate the supported image formats with the *ioctl VIDIOC_ENUM_FMT* ioctl to determine if the device supports MPEG streams.
VID_TYPE_MPEG_ENCODER	-	See above.
VID_TYPE_MJPEG_DECODER	-	See above.
VID_TYPE_MJPEG_ENCODER	-	See above.

The audios field was replaced by capabilities flag V4L2_CAP_AUDIO, indicating *if* the device has any audio inputs or outputs. To determine their number applications can enumerate audio inputs with the *VIDIOC_G_AUDIO* ioctl. The audio ioctls are described in *Audio Inputs and Outputs* .

The maxwidth, maxheight, minwidth and minheight fields were removed. Calling the *VIDIOC_S_FMT* or *VIDIOC_TRY_FMT* ioctl with the desired dimensions returns the closest size possible, taking into account the current video standard, cropping and scaling limitations.

Video Sources

V4L provides the VIDIOCGCHAN and VIDIOCSCHAN ioctl using struct video_channel to enumerate the video inputs of a V4L device. The equivalent V4L2 ioctls are *ioctl VIDIOC_ENUMINPUT* , *VIDIOC_G_INPUT* and *VIDIOC_S_INPUT* using struct *v4l2_input* as discussed in *Video Inputs and Outputs* .

The channel field counting inputs was renamed to index, the video input types were renamed as follows:

struct video_channel type	struct *v4l2_input* type
VIDEO_TYPE_TV	V4L2_INPUT_TYPE_TUNER
VIDEO_TYPE_CAMERA	V4L2_INPUT_TYPE_CAMERA

Unlike the tuners field expressing the number of tuners of this input, V4L2 assumes each video input is connected to at most one tuner. However a tuner can have more than one input, i. e. RF connectors, and a device can have multiple tuners. The index number of the tuner associated with the input, if any, is stored in field tuner of struct *v4l2_input*. Enumeration of tuners is discussed in *Tuners and Modulators* .

The redundant VIDEO_VC_TUNER flag was dropped. Video inputs associated with a tuner are of type V4L2_INPUT_TYPE_TUNER. The VIDEO_VC_AUDIO flag was replaced by the audioset field. V4L2 considers devices with up to 32 audio inputs. Each set bit in the audioset field represents one audio input this video input combines with. For information about audio inputs and how to switch between them see *Audio Inputs and Outputs* .

The norm field describing the supported video standards was replaced by std. The V4L specification mentions a flag VIDEO_VC_NORM indicating whether the standard can be changed. This flag was a later addition together with the norm field and has been removed in the meantime. V4L2 has a similar, albeit more comprehensive approach to video standards, see *Video Standards* for more information.

Tuning

The V4L VIDIOCGTUNER and VIDIOCSTUNER ioctl and struct video_tuner can be used to enumerate the tuners of a V4L TV or radio device. The equivalent V4L2 ioctls are *VIDIOC_G_TUNER* and *VIDIOC_S_TUNER* using struct *v4l2_tuner*. Tuners are covered in *Tuners and Modulators* .

The tuner field counting tuners was renamed to index. The fields name, rangelow and rangehigh remained unchanged.

The VIDEO_TUNER_PAL, VIDEO_TUNER_NTSC and VIDEO_TUNER_SECAM flags indicating the supported video standards were dropped. This information is now contained in the associated struct *v4l2_input*. No replacement exists for the VIDEO_TUNER_NORM flag indicating whether the video standard can be switched. The mode field to select a different video standard was replaced by a whole new set of ioctls and structures described in *Video Standards* . Due to its ubiquity it should be mentioned the BTTV driver supports several standards in addition to the regular VIDEO_MODE_PAL (0), VIDEO_MODE_NTSC, VIDEO_MODE_SECAM and VIDEO_MODE_AUTO (3). Namely N/PAL Argentina, M/PAL, N/PAL, and NTSC Japan with numbers 3-6 (sic).

The VIDEO_TUNER_STEREO_ON flag indicating stereo reception became V4L2_TUNER_SUB_STEREO in field rxsubchans. This field also permits the detection of monaural and bilingual audio, see the definition of struct *v4l2_tuner* for details. Presently no replacement exists for the VIDEO_TUNER_RDS_ON and VIDEO_TUNER_MBS_ON flags.

The VIDEO_TUNER_LOW flag was renamed to V4L2_TUNER_CAP_LOW in the struct *v4l2_tuner* capability field.

The VIDIOCGFREQ and VIDIOCSFREQ ioctl to change the tuner frequency where renamed to *VIDIOC_G_FREQUENCY* and *VIDIOC_S_FREQUENCY* . They take a pointer to a struct *v4l2_frequency* instead of an unsigned long integer.

Image Properties

V4L2 has no equivalent of the VIDIOCGPICT and VIDIOCSPICT ioctl and struct video_picture. The following fields where replaced by V4L2 controls accessible with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* , *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls:

struct video_picture	V4L2 Control ID
brightness	V4L2_CID_BRIGHTNESS
hue	V4L2_CID_HUE
colour	V4L2_CID_SATURATION
contrast	V4L2_CID_CONTRAST
whiteness	V4L2_CID_WHITENESS

The V4L picture controls are assumed to range from 0 to 65535 with no particular reset value. The V4L2 API permits arbitrary limits and defaults which can be queried with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* ioctl. For general information about controls see *User Controls* .

The depth (average number of bits per pixel) of a video image is implied by the selected image format. V4L2 does not explicitly provide such information assuming applications recognizing the format are aware of the image depth and others need not know. The palette field moved into the struct *v4l2_pix_format*:

struct video_picture palette	struct *v4l2_pix_format* pixfmt
VIDEO_PALETTE_GREY	*V4L2_PIX_FMT_GREY*
VIDEO_PALETTE_HI240	*V4L2_PIX_FMT_HI240* [3]
VIDEO_PALETTE_RGB565	*V4L2_PIX_FMT_RGB565*
VIDEO_PALETTE_RGB555	*V4L2_PIX_FMT_RGB555*
VIDEO_PALETTE_RGB24	*V4L2_PIX_FMT_BGR24*
VIDEO_PALETTE_RGB32	*V4L2_PIX_FMT_BGR32* [4]
VIDEO_PALETTE_YUV422	*V4L2_PIX_FMT_YUYV*
VIDEO_PALETTE_YUYV [5]	*V4L2_PIX_FMT_YUYV*
VIDEO_PALETTE_UYVY	*V4L2_PIX_FMT_UYVY*
VIDEO_PALETTE_YUV420	None
VIDEO_PALETTE_YUV411	*V4L2_PIX_FMT_Y41P* [6]
VIDEO_PALETTE_RAW	None [7]
VIDEO_PALETTE_YUV422P	*V4L2_PIX_FMT_YUV422P*
VIDEO_PALETTE_YUV411P	*V4L2_PIX_FMT_YUV411P* [8]
VIDEO_PALETTE_YUV420P	*V4L2_PIX_FMT_YVU420*
VIDEO_PALETTE_YUV410P	*V4L2_PIX_FMT_YVU410*

V4L2 image formats are defined in *Image Formats* . The image format can be selected with the *VIDIOC_S_FMT* ioctl.

Audio

The VIDIOCGAUDIO and VIDIOCSAUDIO ioctl and struct video_audio are used to enumerate the audio inputs of a V4L device. The equivalent V4L2 ioctls are *VIDIOC_G_AUDIO* and *VIDIOC_S_AUDIO* using struct *v4l2_audio* as discussed in *Audio Inputs and Outputs* .

The audio "channel number" field counting audio inputs was renamed to index.

On VIDIOCSAUDIO the mode field selects *one* of the VIDEO_SOUND_MONO, VIDEO_SOUND_STEREO, VIDEO_SOUND_LANG1 or VIDEO_SOUND_LANG2 audio demodulation modes. When the current audio standard is BTSC VIDEO_SOUND_LANG2 refers to SAP and VIDEO_SOUND_LANG1 is meaningless. Also undocumented in the V4L specification, there is no way to query the selected mode. On VIDIOCGAUDIO the driver returns the *actually received* audio programmes in this field. In the V4L2 API this information is stored in the struct *v4l2_tuner* rxsubchans and audmode fields, respectively. See *Tuners and Modulators* for more information on tuners. Related to audio modes struct *v4l2_audio* also reports if this is a mono or stereo input, regardless if the source is a tuner.

[3] This is a custom format used by the BTTV driver, not one of the V4L2 standard formats.

[4] Presumably all V4L RGB formats are little-endian, although some drivers might interpret them according to machine endianness. V4L2 defines little-endian, big-endian and red/blue swapped variants. For details see *RGB Formats* .

[5] VIDEO_PALETTE_YUV422 and VIDEO_PALETTE_YUYV are the same formats. Some V4L drivers respond to one, some to the other.

[6] Not to be confused with V4L2_PIX_FMT_YUV411P, which is a planar format.

[7] V4L explains this as: "RAW capture (BT848)"

[8] Not to be confused with V4L2_PIX_FMT_Y41P, which is a packed format.

The following fields where replaced by V4L2 controls accessible with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* , *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls:

struct video_audio	V4L2 Control ID
volume	V4L2_CID_AUDIO_VOLUME
bass	V4L2_CID_AUDIO_BASS
treble	V4L2_CID_AUDIO_TREBLE
balance	V4L2_CID_AUDIO_BALANCE

To determine which of these controls are supported by a driver V4L provides the flags VIDEO_AUDIO_VOLUME, VIDEO_AUDIO_BASS, VIDEO_AUDIO_TREBLE and VIDEO_AUDIO_BALANCE. In the V4L2 API the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* ioctl reports if the respective control is supported. Accordingly the VIDEO_AUDIO_MUTABLE and VIDEO_AUDIO_MUTE flags where replaced by the boolean V4L2_CID_AUDIO_MUTE control.

All V4L2 controls have a step attribute replacing the struct video_audio step field. The V4L audio controls are assumed to range from 0 to 65535 with no particular reset value. The V4L2 API permits arbitrary limits and defaults which can be queried with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* ioctl. For general information about controls see *User Controls* .

Frame Buffer Overlay

The V4L2 ioctls equivalent to VIDIOCGFBUF and VIDIOCSFBUF are *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF* . The base field of struct video_buffer remained unchanged, except V4L2 defines a flag to indicate non-destructive overlays instead of a NULL pointer. All other fields moved into the struct *v4l2_pix_format* fmt substructure of struct *v4l2_framebuffer*. The depth field was replaced by pixelformat. See *RGB Formats* for a list of RGB formats and their respective color depths.

Instead of the special ioctls VIDIOCGWIN and VIDIOCSWIN V4L2 uses the general-purpose data format negotiation ioctls *VIDIOC_G_FMT* and *VIDIOC_S_FMT* . They take a pointer to a struct *v4l2_format* as argument. Here the win member of the fmt union is used, a struct *v4l2_window*.

The x, y, width and height fields of struct video_window moved into struct *v4l2_rect* substructure w of struct *v4l2_window*. The chromakey, clips, and clipcount fields remained unchanged. Struct video_clip was renamed to struct *v4l2_clip*, also containing a struct *v4l2_rect*, but the semantics are still the same.

The VIDEO_WINDOW_INTERLACE flag was dropped. Instead applications must set the field field to V4L2_FIELD_ANY or V4L2_FIELD_INTERLACED. The VIDEO_WINDOW_CHROMAKEY flag moved into struct *v4l2_framebuffer*, under the new name V4L2_FBUF_FLAG_CHROMAKEY.

In V4L, storing a bitmap pointer in clips and setting clipcount to VIDEO_CLIP_BITMAP (-1) requests bitmap clipping, using a fixed size bitmap of 1024 × 625 bits. Struct *v4l2_window* has a separate bitmap pointer field for this purpose and the bitmap size is determined by w.width and w.height.

The VIDIOCCAPTURE ioctl to enable or disable overlay was renamed to *ioctl VIDIOC_OVERLAY* .

Cropping

To capture only a subsection of the full picture V4L defines the VIDIOCGCAPTURE and VIDIOCSCAPTURE ioctls using struct video_capture. The equivalent V4L2 ioctls are *VIDIOC_G_CROP* and *VIDIOC_S_CROP* using struct *v4l2_crop*, and the related *ioctl VIDIOC_CROPCAP* ioctl. This is a rather complex matter, see *Image Cropping, Insertion and Scaling* for details.

The x, y, width and height fields moved into struct *v4l2_rect* substructure c of struct *v4l2_crop*. The decimation field was dropped. In the V4L2 API the scaling factor is implied by the size of the cropping rectangle and the size of the captured or overlaid image.

The VIDEO_CAPTURE_ODD and VIDEO_CAPTURE_EVEN flags to capture only the odd or even field, respectively, were replaced by V4L2_FIELD_TOP and V4L2_FIELD_BOTTOM in the field named field of struct

v4l2_pix_format and struct *v4l2_window*. These structures are used to select a capture or overlay format with the *VIDIOC_S_FMT* ioctl.

Reading Images, Memory Mapping

Capturing using the read method

There is no essential difference between reading images from a V4L or V4L2 device using the *read()* function, however V4L2 drivers are not required to support this I/O method. Applications can determine if the function is available with the *ioctl VIDIOC_QUERYCAP* ioctl. All V4L2 devices exchanging data with applications must support the *select()* and *poll()* functions.

To select an image format and size, V4L provides the VIDIOCSPICT and VIDIOCSWIN ioctls. V4L2 uses the general-purpose data format negotiation ioctls *VIDIOC_G_FMT* and *VIDIOC_S_FMT* . They take a pointer to a struct *v4l2_format* as argument, here the struct *v4l2_pix_format* named pix of its fmt union is used.

For more information about the V4L2 read interface see *Read/Write* .

Capturing using memory mapping

Applications can read from V4L devices by mapping buffers in device memory, or more often just buffers allocated in DMA-able system memory, into their address space. This avoids the data copying overhead of the read method. V4L2 supports memory mapping as well, with a few differences.

V4L	V4L2
	The image format must be selected before buffers are allocated, with the *VIDIOC_S_FMT* ioctl. When no format is selected the driver may use the last, possibly by another application requested format.
Applications cannot change the number of buffers. The it is built into the driver, unless it has a module option to change the number when the driver module is loaded.	The *ioctl VIDIOC_REQBUFS* ioctl allocates the desired number of buffers, this is a required step in the initialization sequence.
Drivers map all buffers as one contiguous range of memory. The VIDIOCGMBUF ioctl is available to query the number of buffers, the offset of each buffer from the start of the virtual file, and the overall amount of memory used, which can be used as arguments for the *mmap()* function.	Buffers are individually mapped. The offset and size of each buffer can be determined with the *ioctl VIDIOC_QUERYBUF* ioctl.
The VIDIOCMCAPTURE ioctl prepares a buffer for capturing. It also determines the image format for this buffer. The ioctl returns immediately, eventually with an EAGAIN error code if no video signal had been detected. When the driver supports more than one buffer applications can call the ioctl multiple times and thus have multiple outstanding capture requests. The VIDIOCSYNC ioctl suspends execution until a particular buffer has been filled.	Drivers maintain an incoming and outgoing queue. *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* enqueues any empty buffer into the incoming queue. Filled buffers are dequeued from the outgoing queue with the *VIDIOC_DQBUF* ioctl. To wait until filled buffers become available this function, *select()* or *poll()* can be used. The *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* ioctl must be called once after enqueuing one or more buffers to start capturing. Its counterpart *VIDIOC_STREAMOFF* stops capturing and dequeues all buffers from both queues. Applications can query the signal status, if known, with the *ioctl VIDIOC_ENUMINPUT* ioctl.

For a more in-depth discussion of memory mapping and examples, see *Streaming I/O (Memory Mapping)* .

Reading Raw VBI Data

Originally the V4L API did not specify a raw VBI capture interface, only the device file /dev/vbi was reserved for this purpose. The only driver supporting this interface was the BTTV driver, de-facto defining the V4L VBI interface. Reading from the device yields a raw VBI image with the following parameters:

struct *v4l2_vbi_format*	V4L, BTTV driver
sampling_rate	28636363 Hz NTSC (or any other 525-line standard); 35468950 Hz PAL and SECAM (625-line standards)
offset	?
samples_per_line	2048
sample_format	V4L2_PIX_FMT_GREY. The last four bytes (a machine endianness integer) contain a frame counter.
start[]	10, 273 NTSC; 22, 335 PAL and SECAM
count[]	16, 16 [9]
flags	0

Undocumented in the V4L specification, in Linux 2.3 the VIDIOCGVBIFMT and VIDIOCSVBIFMT ioctls using struct vbi_format were added to determine the VBI image parameters. These ioctls are only partially compatible with the V4L2 VBI interface specified in *Raw VBI Data Interface* .

An offset field does not exist, sample_format is supposed to be VIDEO_PALETTE_RAW, equivalent to V4L2_PIX_FMT_GREY. The remaining fields are probably equivalent to struct *v4l2_vbi_format*.

Apparently only the Zoran (ZR 36120) driver implements these ioctls. The semantics differ from those specified for V4L2 in two ways. The parameters are reset on *open()* and VIDIOCSVBIFMT always returns an EINVAL error code if the parameters are invalid.

Miscellaneous

V4L2 has no equivalent of the VIDIOCGUNIT ioctl. Applications can find the VBI device associated with a video capture device (or vice versa) by reopening the device and requesting VBI data. For details see *Opening and Closing Devices* .

No replacement exists for VIDIOCKEY, and the V4L functions for microcode programming. A new interface for MPEG compression and playback devices is documented in *Extended Controls* .

Changes of the V4L2 API

Soon after the V4L API was added to the kernel it was criticised as too inflexible. In August 1998 Bill Dirks proposed a number of improvements and began to work on documentation, example drivers and applications. With the help of other volunteers this eventually became the V4L2 API, not just an extension but a replacement for the V4L API. However it took another four years and two stable kernel releases until the new API was finally accepted for inclusion into the kernel in its present form.

Early Versions

1998-08-20: First version.

1998-08-27: The *select()* function was introduced.

1998-09-10: New video standard interface.

1998-09-18: The VIDIOC_NONCAP ioctl was replaced by the otherwise meaningless O_TRUNC *open()* flag, and the aliases O_NONCAP and O_NOIO were defined. Applications can set this flag if they intend to access

[9] Old driver versions used different values, eventually the custom BTTV_VBISIZE ioctl was added to query the correct values.

controls only, as opposed to capture applications which need exclusive access. The VIDEO_STD_XXX identifiers are now ordinals instead of flags, and the video_std_construct() helper function takes id and transmission arguments.

1998-09-28: Revamped video standard. Made video controls individually enumerable.

1998-10-02: The id field was removed from struct struct video_standard and the color subcarrier fields were renamed. The *ioctl VIDIOC_QUERYSTD* ioctl was renamed to *ioctl VIDIOC_ENUMSTD* , *VIDIOC_G_INPUT* to *ioctl VIDIOC_ENUMINPUT* . A first draft of the Codec API was released.

1998-11-08: Many minor changes. Most symbols have been renamed. Some material changes to struct *v4l2_capability*.

1998-11-12: The read/write directon of some ioctls was misdefined.

1998-11-14: V4L2_PIX_FMT_RGB24 changed to V4L2_PIX_FMT_BGR24, and V4L2_PIX_FMT_RGB32 changed to V4L2_PIX_FMT_BGR32. Audio controls are now accessible with the *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls under names starting with V4L2_CID_AUDIO. The V4L2_MAJOR define was removed from videodev.h since it was only used once in the videodev kernel module. The YUV422 and YUV411 planar image formats were added.

1998-11-28: A few ioctl symbols changed. Interfaces for codecs and video output devices were added.

1999-01-14: A raw VBI capture interface was added.

1999-01-19: The VIDIOC_NEXTBUF ioctl was removed.

V4L2 Version 0.16 1999-01-31

1999-01-27: There is now one QBUF ioctl, VIDIOC_QWBUF and VIDIOC_QRBUF are gone. VIDIOC_QBUF takes a v4l2_buffer as a parameter. Added digital zoom (cropping) controls.

V4L2 Version 0.18 1999-03-16

Added a v4l to V4L2 ioctl compatibility layer to videodev.c. Driver writers, this changes how you implement your ioctl handler. See the Driver Writer's Guide. Added some more control id codes.

V4L2 Version 0.19 1999-06-05

1999-03-18: Fill in the category and catname fields of v4l2_queryctrl objects before passing them to the driver. Required a minor change to the VIDIOC_QUERYCTRL handlers in the sample drivers.

1999-03-31: Better compatibility for v4l memory capture ioctls. Requires changes to drivers to fully support new compatibility features, see Driver Writer's Guide and v4l2cap.c. Added new control IDs: V4L2_CID_HFLIP, _VFLIP. Changed V4L2_PIX_FMT_YUV422P to _YUV422P, and _YUV411P to _YUV411P.

1999-04-04: Added a few more control IDs.

1999-04-07: Added the button control type.

1999-05-02: Fixed a typo in videodev.h, and added the V4L2_CTRL_FLAG_GRAYED (later V4L2_CTRL_FLAG_GRABBED) flag.

1999-05-20: Definition of VIDIOC_G_CTRL was wrong causing a malfunction of this ioctl.

1999-06-05: Changed the value of V4L2_CID_WHITENESS.

V4L2 Version 0.20 (1999-09-10)

Version 0.20 introduced a number of changes which were *not backward compatible* with 0.19 and earlier versions. Purpose of these changes was to simplify the API, while making it more extensible and following common Linux driver API conventions.

1. Some typos in V4L2_FMT_FLAG symbols were fixed. struct *v4l2_clip* was changed for compatibility with v4l. (1999-08-30)

2. V4L2_TUNER_SUB_LANG1 was added. (1999-09-05)

3. All ioctl() commands that used an integer argument now take a pointer to an integer. Where it makes sense, ioctls will return the actual new value in the integer pointed to by the argument, a common convention in the V4L2 API. The affected ioctls are: VIDIOC_PREVIEW, VIDIOC_STREAMON, VIDIOC_STREAMOFF, VIDIOC_S_FREQ, VIDIOC_S_INPUT, VIDIOC_S_OUTPUT, VIDIOC_S_EFFECT. For example

   ```
   err = ioctl (fd, VIDIOC_XXX, V4L2_XXX);
   ```

   becomes

   ```
   int a = V4L2_XXX; err = ioctl(fd, VIDIOC_XXX, &a);
   ```

4. All the different get- and set-format commands were swept into one *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl taking a union and a type field selecting the union member as parameter. Purpose is to simplify the API by eliminating several ioctls and to allow new and driver private data streams without adding new ioctls.

   This change obsoletes the following ioctls: VIDIOC_S_INFMT, VIDIOC_G_INFMT, VIDIOC_S_OUTFMT, VIDIOC_G_OUTFMT, VIDIOC_S_VBIFMT and VIDIOC_G_VBIFMT. The image format structure struct *v4l2_format* was renamed to struct *v4l2_pix_format*, while struct *v4l2_format* is now the enveloping structure for all format negotiations.

5. Similar to the changes above, the VIDIOC_G_PARM and VIDIOC_S_PARM ioctls were merged with VIDIOC_G_OUTPARM and VIDIOC_S_OUTPARM. A type field in the new struct *v4l2_streamparm* selects the respective union member.

   This change obsoletes the VIDIOC_G_OUTPARM and VIDIOC_S_OUTPARM ioctls.

6. Control enumeration was simplified, and two new control flags were introduced and one dropped. The catname field was replaced by a group field.

   Drivers can now flag unsupported and temporarily unavailable controls with V4L2_CTRL_FLAG_DISABLED and V4L2_CTRL_FLAG_GRABBED respectively. The group name indicates a possibly narrower classification than the category. In other words, there may be multiple groups within a category. Controls within a group would typically be drawn within a group box. Controls in different categories might have a greater separation, or may even appear in separate windows.

7. The struct *v4l2_buffer* timestamp was changed to a 64 bit integer, containing the sampling or output time of the frame in nanoseconds. Additionally timestamps will be in absolute system time, not starting from zero at the beginning of a stream. The data type name for timestamps is stamp_t, defined as a signed 64-bit integer. Output devices should not send a buffer out until the time in the timestamp field has arrived. I would like to follow SGI's lead, and adopt a multimedia timestamping system like their UST (Unadjusted System Time). See http://web.archive.org/web/*/http://reality.sgi.com /cpirazzi_engr/lg/time/intro.html. UST uses timestamps that are 64-bit signed integers (not struct timeval's) and given in nanosecond units. The UST clock starts at zero when the system is booted and runs continuously and uniformly. It takes a little over 292 years for UST to overflow. There is no way to set the UST clock. The regular Linux time-of-day clock can be changed periodically, which would cause errors if it were being used for timestamping a multimedia stream. A real UST style clock will require some support in the kernel that is not there yet. But in anticipation, I will change the timestamp field to a 64-bit integer, and I will change the v4l2_masterclock_gettime() function (used only by drivers) to return a 64-bit integer.

8. A sequence field was added to struct *v4l2_buffer*. The sequence field counts captured frames, it is ignored by output devices. When a capture driver drops a frame, the sequence number of that frame is skipped.

V4L2 Version 0.20 incremental changes

1999-12-23: In struct *v4l2_vbi_format* the reserved1 field became offset. Previously drivers were required to clear the reserved1 field.

2000-01-13: The V4L2_FMT_FLAG_NOT_INTERLACED flag was added.

2000-07-31: The linux/poll.h header is now included by videodev.h for compatibility with the original videodev.h file.

2000-11-20: V4L2_TYPE_VBI_OUTPUT and V4L2_PIX_FMT_Y41P were added.

2000-11-25: V4L2_TYPE_VBI_INPUT was added.

2000-12-04: A couple typos in symbol names were fixed.

2001-01-18: To avoid namespace conflicts the fourcc macro defined in the videodev.h header file was renamed to v4l2_fourcc.

2001-01-25: A possible driver-level compatibility problem between the videodev.h file in Linux 2.4.0 and the videodev.h file included in the videodevX patch was fixed. Users of an earlier version of videodevX on Linux 2.4.0 should recompile their V4L and V4L2 drivers.

2001-01-26: A possible kernel-level incompatibility between the videodev.h file in the videodevX patch and the videodev.h file in Linux 2.2.x with devfs patches applied was fixed.

2001-03-02: Certain V4L ioctls which pass data in both direction although they are defined with read-only parameter, did not work correctly through the backward compatibility layer. [Solution?]

2001-04-13: Big endian 16-bit RGB formats were added.

2001-09-17: New YUV formats and the *VIDIOC_G_FREQUENCY* and *VIDIOC_S_FREQUENCY* ioctls were added. (The old VIDIOC_G_FREQ and VIDIOC_S_FREQ ioctls did not take multiple tuners into account.)

2000-09-18: V4L2_BUF_TYPE_VBI was added. This may *break compatibility* as the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls may fail now if the struct struct v4l2_fmt type field does not contain V4L2_BUF_TYPE_VBI. In the documentation of the struct *v4l2_vbi_format* offset field the ambiguous phrase “rising edge” was changed to “leading edge”.

V4L2 Version 0.20 2000-11-23

A number of changes were made to the raw VBI interface.

1. Figures clarifying the line numbering scheme were added to the V4L2 API specification. The start[0] and start[1] fields no longer count line numbers beginning at zero. Rationale: a) The previous definition was unclear. b) The start[] values are ordinal numbers. c) There is no point in inventing a new line numbering scheme. We now use line number as defined by ITU-R, period. Compatibility: Add one to the start values. Applications depending on the previous semantics may not function correctly.
2. The restriction “count[0] > 0 and count[1] > 0” has been relaxed to “(count[0] + count[1]) > 0”. Rationale: Drivers may allocate resources at scan line granularity and some data services are transmitted only on the first field. The comment that both count values will usually be equal is misleading and pointless and has been removed. This change *breaks compatibility* with earlier versions: Drivers may return EINVAL, applications may not function correctly.
3. Drivers are again permitted to return negative (unknown) start values as proposed earlier. Why this feature was dropped is unclear. This change may *break compatibility* with applications depending on the start values being positive. The use of EBUSY and EINVAL error codes with the *VIDIOC_S_FMT*

ioctl was clarified. The EBUSY error code was finally documented, and the reserved2 field which was previously mentioned only in the videodev.h header file.

4. New buffer types V4L2_TYPE_VBI_INPUT and V4L2_TYPE_VBI_OUTPUT were added. The former is an alias for the old V4L2_TYPE_VBI, the latter was missing in the videodev.h file.

V4L2 Version 0.20 2002-07-25

Added sliced VBI interface proposal.

V4L2 in Linux 2.5.46, 2002-10

Around October-November 2002, prior to an announced feature freeze of Linux 2.5, the API was revised, drawing from experience with V4L2 0.20. This unnamed version was finally merged into Linux 2.5.46.

1. As specified in *Related Devices* , drivers must make related device functions available under all minor device numbers.
2. The *open()* function requires access mode O_RDWR regardless of the device type. All V4L2 drivers exchanging data with applications must support the O_NONBLOCK flag. The O_NOIO flag, a V4L2 symbol which aliased the meaningless O_TRUNC to indicate accesses without data exchange (panel applications) was dropped. Drivers must stay in "panel mode" until the application attempts to initiate a data exchange, see *Opening and Closing Devices* .
3. The struct *v4l2_capability* changed dramatically. Note that also the size of the structure changed, which is encoded in the ioctl request code, thus older V4L2 devices will respond with an EINVAL error code to the new *ioctl VIDIOC_QUERYCAP* ioctl.

   There are new fields to identify the driver, a new RDS device function V4L2_CAP_RDS_CAPTURE, the V4L2_CAP_AUDIO flag indicates if the device has any audio connectors, another I/O capability V4L2_CAP_ASYNCIO can be flagged. In response to these changes the type field became a bit set and was merged into the flags field. V4L2_FLAG_TUNER was renamed to V4L2_CAP_TUNER, V4L2_CAP_VIDEO_OVERLAY replaced V4L2_FLAG_PREVIEW and V4L2_CAP_VBI_CAPTURE and V4L2_CAP_VBI_OUTPUT replaced V4L2_FLAG_DATA_SERVICE. V4L2_FLAG_READ and V4L2_FLAG_WRITE were merged into V4L2_CAP_READWRITE.

   The redundant fields inputs, outputs and audios were removed. These properties can be determined as described in *Video Inputs and Outputs* and *Audio Inputs and Outputs* .

   The somewhat volatile and therefore barely useful fields maxwidth, maxheight, minwidth, minheight, maxframerate were removed. This information is available as described in *Data Formats* and *Video Standards* .

   V4L2_FLAG_SELECT was removed. We believe the select() function is important enough to require support of it in all V4L2 drivers exchanging data with applications. The redundant V4L2_FLAG_MONOCHROME flag was removed, this information is available as described in *Data Formats* .
4. In struct *v4l2_input* the assoc_audio field and the capability field and its only flag V4L2_INPUT_CAP_AUDIO was replaced by the new audioset field. Instead of linking one video input to one audio input this field reports all audio inputs this video input combines with.

   New fields are tuner (reversing the former link from tuners to video inputs), std and status.

   Accordingly struct *v4l2_output* lost its capability and assoc_audio fields. audioset, modulator and std where added instead.
5. The struct *v4l2_audio* field audio was renamed to index, for consistency with other structures. A new capability flag V4L2_AUDCAP_STEREO was added to indicated if the audio input in question supports stereo sound. V4L2_AUDCAP_EFFECTS and the corresponding V4L2_AUDMODE flags where removed. This can be easily implemented using controls. (However the same applies to AVL which is still there.)

Again for consistency the struct *v4l2_audioout* field audio was renamed to index.

6. The struct *v4l2_tuner* input field was replaced by an index field, permitting devices with multiple tuners. The link between video inputs and tuners is now reversed, inputs point to their tuner. The std substructure became a simple set (more about this below) and moved into struct *v4l2_input*. A type field was added.

   Accordingly in struct *v4l2_modulator* the output was replaced by an index field.

   In struct *v4l2_frequency* the port field was replaced by a tuner field containing the respective tuner or modulator index number. A tuner type field was added and the reserved field became larger for future extensions (satellite tuners in particular).

7. The idea of completely transparent video standards was dropped. Experience showed that applications must be able to work with video standards beyond presenting the user a menu. Instead of enumerating supported standards with an ioctl applications can now refer to standards by *v4l2_std_id* and symbols defined in the videodev2.h header file. For details see *Video Standards* . The *VIDIOC_G_STD* and *VIDIOC_S_STD* now take a pointer to this type as argument. *ioctl VIDIOC_QUERYSTD* was added to autodetect the received standard, if the hardware has this capability. In struct *v4l2_standard* an index field was added for *ioctl VIDIOC_ENUMSTD* . A *v4l2_std_id* field named id was added as machine readable identifier, also replacing the transmission field. The misleading framerate field was renamed to frameperiod. The now obsolete colorstandard information, originally needed to distguish between variations of standards, were removed.

   Struct v4l2_enumstd ceased to be. *ioctl VIDIOC_ENUMSTD* now takes a pointer to a struct *v4l2_standard* directly. The information which standards are supported by a particular video input or output moved into struct *v4l2_input* and struct *v4l2_output* fields named std, respectively.

8. The struct *v4l2_queryctrl* fields category and group did not catch on and/or were not implemented as expected and therefore removed.

9. The *VIDIOC_TRY_FMT* ioctl was added to negotiate data formats as with *VIDIOC_S_FMT* , but without the overhead of programming the hardware and regardless of I/O in progress.

   In struct *v4l2_format* the fmt union was extended to contain struct *v4l2_window*. All image format negotiations are now possible with VIDIOC_G_FMT, VIDIOC_S_FMT and VIDIOC_TRY_FMT; ioctl. The VIDIOC_G_WIN and VIDIOC_S_WIN ioctls to prepare for a video overlay were removed. The type field changed to type enum *v4l2_buf_type* and the buffer type names changed as follows.

Old defines	enum *v4l2_buf_type*
V4L2_BUF_TYPE_CAPTURE	V4L2_BUF_TYPE_VIDEO_CAPTURE
V4L2_BUF_TYPE_CODECIN	Omitted for now
V4L2_BUF_TYPE_CODECOUT	Omitted for now
V4L2_BUF_TYPE_EFFECTSIN	Omitted for now
V4L2_BUF_TYPE_EFFECTSIN2	Omitted for now
V4L2_BUF_TYPE_EFFECTSOUT	Omitted for now
V4L2_BUF_TYPE_VIDEOOUT	V4L2_BUF_TYPE_VIDEO_OUTPUT
-	V4L2_BUF_TYPE_VIDEO_OVERLAY
-	V4L2_BUF_TYPE_VBI_CAPTURE
-	V4L2_BUF_TYPE_VBI_OUTPUT
-	V4L2_BUF_TYPE_SLICED_VBI_CAPTURE
-	V4L2_BUF_TYPE_SLICED_VBI_OUTPUT
V4L2_BUF_TYPE_PRIVATE_BASE	V4L2_BUF_TYPE_PRIVATE (but this is deprecated)

10. In struct *v4l2_fmtdesc* a enum *v4l2_buf_type* field named type was added as in struct *v4l2_format*. The VIDIOC_ENUM_FBUFFMT ioctl is no longer needed and was removed. These calls can be replaced by *ioctl VIDIOC_ENUM_FMT* with type V4L2_BUF_TYPE_VIDEO_OVERLAY.

11. In struct *v4l2_pix_format* the depth field was removed, assuming applications which recognize the format by its four-character-code already know the color depth, and others do not care about it. The same rationale lead to the removal of the V4L2_FMT_FLAG_COMPRESSED flag. The V4L2_FMT_FLAG_SWCONVECOMPRESSED flag was removed because drivers are not supposed to convert images in kernel space. A user library of conversion functions should be provided instead. The

V4L2_FMT_FLAG_BYTESPERLINE flag was redundant. Applications can set the bytesperline field to zero to get a reasonable default. Since the remaining flags were replaced as well, the flags field itself was removed.

The interlace flags were replaced by a enum *v4l2_field* value in a newly added field field.

Old flag	enum *v4l2_field*
V4L2_FMT_FLAG_NOT_INTERLACED	?
V4L2_FMT_FLAG_INTERLACED = V4L2_FMT_FLAG_COMBINED	V4L2_FIELD_INTERLACED
V4L2_FMT_FLAG_TOPFIELD = V4L2_FMT_FLAG_ODDFIELD	V4L2_FIELD_TOP
V4L2_FMT_FLAG_BOTFIELD = V4L2_FMT_FLAG_EVENFIELD	V4L2_FIELD_BOTTOM
-	V4L2_FIELD_SEQ_TB
-	V4L2_FIELD_SEQ_BT
-	V4L2_FIELD_ALTERNATE

The color space flags were replaced by a enum *v4l2_colorspace* value in a newly added colorspace field, where one of V4L2_COLORSPACE_SMPTE170M, V4L2_COLORSPACE_BT878, V4L2_COLORSPACE_470_SYSTEM_M or V4L2_COLORSPACE_470_SYSTEM_BG replaces V4L2_FMT_CS_601YUV.

12. In struct *v4l2_requestbuffers* the type field was properly defined as enum *v4l2_buf_type*. Buffer types changed as mentioned above. A new memory field of type enum *v4l2_memory* was added to distinguish between I/O methods using buffers allocated by the driver or the application. See *Input/Output* for details.

13. In struct *v4l2_buffer* the type field was properly defined as enum *v4l2_buf_type*. Buffer types changed as mentioned above. A field field of type enum *v4l2_field* was added to indicate if a buffer contains a top or bottom field. The old field flags were removed. Since no unadjusted system time clock was added to the kernel as planned, the timestamp field changed back from type stamp_t, an unsigned 64 bit integer expressing the sample time in nanoseconds, to struct timeval. With the addition of a second memory mapping method the offset field moved into union m, and a new memory field of type enum *v4l2_memory* was added to distinguish between I/O methods. See *Input/Output* for details.

    The V4L2_BUF_REQ_CONTIG flag was used by the V4L compatibility layer, after changes to this code it was no longer needed. The V4L2_BUF_ATTR_DEVICEMEM flag would indicate if the buffer was indeed allocated in device memory rather than DMA-able system memory. It was barely useful and so was removed.

14. In struct *v4l2_framebuffer* the base[3] array anticipating double- and triple-buffering in off-screen video memory, however without defining a synchronization mechanism, was replaced by a single pointer. The V4L2_FBUF_CAP_SCALEUP and V4L2_FBUF_CAP_SCALEDOWN flags were removed. Applications can determine this capability more accurately using the new cropping and scaling interface. The V4L2_FBUF_CAP_CLIPPING flag was replaced by V4L2_FBUF_CAP_LIST_CLIPPING and V4L2_FBUF_CAP_BITMAP_CLIPPING.

15. In struct *v4l2_clip* the x, y, width and height field moved into a c substructure of type struct *v4l2_rect*. The x and y fields were renamed to left and top, i. e. offsets to a context dependent origin.

16. In struct *v4l2_window* the x, y, width and height field moved into a w substructure as above. A field field of type *v4l2_field* was added to distinguish between field and frame (interlaced) overlay.

17. The digital zoom interface, including struct struct v4l2_zoomcap, struct struct v4l2_zoom, V4L2_ZOOM_NONCAP and V4L2_ZOOM_WHILESTREAMING was replaced by a new cropping and scaling interface. The previously unused struct struct *v4l2_cropcap* and struct *v4l2_crop* where redefined for this purpose. See *Image Cropping, Insertion and Scaling* for details.

18. In struct *v4l2_vbi_format* the SAMPLE_FORMAT field now contains a four-character-code as used to identify video image formats and V4L2_PIX_FMT_GREY replaces the V4L2_VBI_SF_UBYTE define. The reserved field was extended.

19. In struct *v4l2_captureparm* the type of the timeperframe field changed from unsigned long to struct *v4l2_fract*. This allows the accurate expression of multiples of the NTSC-M frame rate 30000 / 1001.

A new field readbuffers was added to control the driver behaviour in read I/O mode.

Similar changes were made to struct *v4l2_outputparm*.

20. The struct v4l2_performance and VIDIOC_G_PERF ioctl were dropped. Except when using the *read/write I/O method* , which is limited anyway, this information is already available to applications.
21. The example transformation from RGB to YCbCr color space in the old V4L2 documentation was inaccurate, this has been corrected in *Image Formats* .

V4L2 2003-06-19

1. A new capability flag V4L2_CAP_RADIO was added for radio devices. Prior to this change radio devices would identify solely by having exactly one tuner whose type field reads V4L2_TUNER_RADIO.
2. An optional driver access priority mechanism was added, see *Application Priority* for details.
3. The audio input and output interface was found to be incomplete.

   Previously the *VIDIOC_G_AUDIO* ioctl would enumerate the available audio inputs. An ioctl to determine the current audio input, if more than one combines with the current video input, did not exist. So VIDIOC_G_AUDIO was renamed to VIDIOC_G_AUDIO_OLD, this ioctl was removed on Kernel 2.6.39. The *ioctl VIDIOC_ENUMAUDIO* ioctl was added to enumerate audio inputs, while *VIDIOC_G_AUDIO* now reports the current audio input.

   The same changes were made to *VIDIOC_G_AUDOUT* and *VIDIOC_ENUMAUDOUT* .

   Until further the “videodev” module will automatically translate between the old and new ioctls, but drivers and applications must be updated to successfully compile again.
4. The *ioctl VIDIOC_OVERLAY* ioctl was incorrectly defined with write-read parameter. It was changed to write-only, while the write-read version was renamed to VIDIOC_OVERLAY_OLD. The old ioctl was removed on Kernel 2.6.39. Until further the “videodev” kernel module will automatically translate to the new version, so drivers must be recompiled, but not applications.
5. *Video Overlay Interface* incorrectly stated that clipping rectangles define regions where the video can be seen. Correct is that clipping rectangles define regions where *no* video shall be displayed and so the graphics surface can be seen.
6. The *VIDIOC_S_PARM* and *VIDIOC_S_CTRL* ioctls were defined with write-only parameter, inconsistent with other ioctls modifying their argument. They were changed to write-read, while a _OLD suffix was added to the write-only versions. The old ioctls were removed on Kernel 2.6.39. Drivers and applications assuming a constant parameter need an update.

V4L2 2003-11-05

1. In *RGB Formats* the following pixel formats were incorrectly transferred from Bill Dirks’ V4L2 specification. Descriptions below refer to bytes in memory, in ascending address order.

Symbol	In this document prior to revision 0.5	Corrected
V4L2_PIX_FMT_RGB24	B, G, R	R, G, B
V4L2_PIX_FMT_BGR24	R, G, B	B, G, R
V4L2_PIX_FMT_RGB32	B, G, R, X	R, G, B, X
V4L2_PIX_FMT_BGR32	R, G, B, X	B, G, R, X

   The V4L2_PIX_FMT_BGR24 example was always correct.

   In *Image Properties* the mapping of the V4L VIDEO_PALETTE_RGB24 and VIDEO_PALETTE_RGB32 formats to V4L2 pixel formats was accordingly corrected.
2. Unrelated to the fixes above, drivers may still interpret some V4L2 RGB pixel formats differently. These issues have yet to be addressed, for details see *RGB Formats* .

V4L2 in Linux 2.6.6, 2004-05-09

1. The *ioctl VIDIOC_CROPCAP* ioctl was incorrectly defined with read-only parameter. It is now defined as write-read ioctl, while the read-only version was renamed to VIDIOC_CROPCAP_OLD. The old ioctl was removed on Kernel 2.6.39.

V4L2 in Linux 2.6.8

1. A new field input (former reserved[0]) was added to the struct *v4l2_buffer* structure. Purpose of this field is to alternate between video inputs (e. g. cameras) in step with the video capturing process. This function must be enabled with the new V4L2_BUF_FLAG_INPUT flag. The flags field is no longer read-only.

V4L2 spec erratum 2004-08-01

1. The return value of the *V4L2 open()* function was incorrectly documented.
2. Audio output ioctls end in -AUDOUT, not -AUDIOOUT.
3. In the Current Audio Input example the VIDIOC_G_AUDIO ioctl took the wrong argument.
4. The documentation of the *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* and *VIDIOC_DQBUF* ioctls did not mention the struct *v4l2_buffer* memory field. It was also missing from examples. Also on the VIDIOC_DQBUF page the EIO error code was not documented.

V4L2 in Linux 2.6.14

1. A new sliced VBI interface was added. It is documented in *Sliced VBI Data Interface* and replaces the interface first proposed in V4L2 specification 0.8.

V4L2 in Linux 2.6.15

1. The *ioctl VIDIOC_LOG_STATUS* ioctl was added.
2. New video standards V4L2_STD_NTSC_443, V4L2_STD_SECAM_LC, V4L2_STD_SECAM_DK (a set of SECAM D, K and K1), and V4L2_STD_ATSC (a set of V4L2_STD_ATSC_8_VSB and V4L2_STD_ATSC_16_VSB) were defined. Note the V4L2_STD_525_60 set now includes V4L2_STD_NTSC_443. See also *typedef v4l2_std_id* .
3. The VIDIOC_G_COMP and VIDIOC_S_COMP ioctl were renamed to VIDIOC_G_MPEGCOMP and VIDIOC_S_MPEGCOMP respectively. Their argument was replaced by a struct v4l2_mpeg_compression pointer. (The VIDIOC_G_MPEGCOMP and VIDIOC_S_MPEGCOMP ioctls where removed in Linux 2.6.25.)

V4L2 spec erratum 2005-11-27

The capture example in *Video Capture Example* called the *VIDIOC_S_CROP* ioctl without checking if cropping is supported. In the video standard selection example in *Video Standards* the *VIDIOC_S_STD* call used the wrong argument type.

V4L2 spec erratum 2006-01-10

1. The V4L2_IN_ST_COLOR_KILL flag in struct *v4l2_input* not only indicates if the color killer is enabled, but also if it is active. (The color killer disables color decoding when it detects no color in the video signal to improve the image quality.)

2. *VIDIOC_S_PARM* is a write-read ioctl, not write-only as stated on its reference page. The ioctl changed in 2003 as noted above.

V4L2 spec erratum 2006-02-03

1. In struct *v4l2_captureparm* and struct *v4l2_outputparm* the `timeperframe` field gives the time in seconds, not microseconds.

V4L2 spec erratum 2006-02-04

1. The `clips` field in struct *v4l2_window* must point to an array of struct *v4l2_clip*, not a linked list, because drivers ignore the struct struct *v4l2_clip*. `next` pointer.

V4L2 in Linux 2.6.17

1. New video standard macros were added: `V4L2_STD_NTSC_M_KR` (NTSC M South Korea), and the sets `V4L2_STD_MN`, `V4L2_STD_B`, `V4L2_STD_GH` and `V4L2_STD_DK`. The `V4L2_STD_NTSC` and `V4L2_STD_SECAM` sets now include `V4L2_STD_NTSC_M_KR` and `V4L2_STD_SECAM_LC` respectively.
2. A new `V4L2_TUNER_MODE_LANG1_LANG2` was defined to record both languages of a bilingual program. The use of `V4L2_TUNER_MODE_STEREO` for this purpose is deprecated now. See the *VIDIOC_G_TUNER* section for details.

V4L2 spec erratum 2006-09-23 (Draft 0.15)

1. In various places `V4L2_BUF_TYPE_SLICED_VBI_CAPTURE` and `V4L2_BUF_TYPE_SLICED_VBI_OUTPUT` of the sliced VBI interface were not mentioned along with other buffer types.
2. In *VIDIOC_G_AUDIO* it was clarified that the struct *v4l2_audio* `mode` field is a flags field.
3. *ioctl VIDIOC_QUERYCAP* did not mention the sliced VBI and radio capability flags.
4. In *VIDIOC_G_FREQUENCY* it was clarified that applications must initialize the tuner `type` field of struct *v4l2_frequency* before calling *VIDIOC_S_FREQUENCY* .
5. The `reserved` array in struct *v4l2_requestbuffers* has 2 elements, not 32.
6. In *Video Output Interface* and *Raw VBI Data Interface* the device file names `/dev/vout` which never caught on were replaced by `/dev/video`.
7. With Linux 2.6.15 the possible range for VBI device minor numbers was extended from 224-239 to 224-255. Accordingly device file names `/dev/vbi0` to `/dev/vbi31` are possible now.

V4L2 in Linux 2.6.18

1. New ioctls *VIDIOC_G_EXT_CTRLS* , *VIDIOC_S_EXT_CTRLS* and *VIDIOC_TRY_EXT_CTRLS* were added, a flag to skip unsupported controls with *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* , new control types `V4L2_CTRL_TYPE_INTEGER64` and `V4L2_CTRL_TYPE_CTRL_CLASS` (*v4l2_ctrl_type*), and new control flags `V4L2_CTRL_FLAG_READ_ONLY`, `V4L2_CTRL_FLAG_UPDATE`, `V4L2_CTRL_FLAG_INACTIVE` and `V4L2_CTRL_FLAG_SLIDER` (*Control Flags*). See *Extended Controls* for details.

V4L2 in Linux 2.6.19

1. In struct *v4l2_sliced_vbi_cap* a buffer type field was added replacing a reserved field. Note on architectures where the size of enum types differs from int types the size of the structure changed. The *VIDIOC_G_SLICED_VBI_CAP* ioctl was redefined from being read-only to write-read. Applications must initialize the type field and clear the reserved fields now. These changes may *break the compatibility* with older drivers and applications.
2. The ioctls *ioctl VIDIOC_ENUM_FRAMESIZES* and *ioctl VIDIOC_ENUM_FRAMEINTERVALS* were added.
3. A new pixel format V4L2_PIX_FMT_RGB444 (*Packed RGB Image Formats*) was added.

V4L2 spec erratum 2006-10-12 (Draft 0.17)

1. V4L2_PIX_FMT_HM12 (*Reserved Image Formats*) is a YUV 4:2:0, not 4:2:2 format.

V4L2 in Linux 2.6.21

1. The videodev2.h header file is now dual licensed under GNU General Public License version two or later, and under a 3-clause BSD-style license.

V4L2 in Linux 2.6.22

1. Two new field orders V4L2_FIELD_INTERLACED_TB and V4L2_FIELD_INTERLACED_BT were added. See *v4l2_field* for details.
2. Three new clipping/blending methods with a global or straight or inverted local alpha value were added to the video overlay interface. See the description of the *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF* ioctls for details.

   A new global_alpha field was added to *v4l2_window*, extending the structure. This may *break compatibility* with applications using a struct struct *v4l2_window* directly. However the *VIDIOC_G/S/TRY_FMT* ioctls, which take a pointer to a *v4l2_format* parent structure with padding bytes at the end, are not affected.
3. The format of the chromakey field in struct *v4l2_window* changed from “host order RGB32” to a pixel value in the same format as the framebuffer. This may *break compatibility* with existing applications. Drivers supporting the “host order RGB32” format are not known.

V4L2 in Linux 2.6.24

1. The pixel formats V4L2_PIX_FMT_PAL8, V4L2_PIX_FMT_YUV444, V4L2_PIX_FMT_YUV555, V4L2_PIX_FMT_YUV565 and V4L2_PIX_FMT_YUV32 were added.

V4L2 in Linux 2.6.25

1. The pixel formats *V4L2_PIX_FMT_Y16* and *V4L2_PIX_FMT_SBGGR16* were added.
2. New *controls* V4L2_CID_POWER_LINE_FREQUENCY, V4L2_CID_HUE_AUTO, V4L2_CID_WHITE_BALANCE_TEMPERATURE, V4L2_CID_SHARPNESS and V4L2_CID_BACKLIGHT_COMPENSATION were added. The controls V4L2_CID_BLACK_LEVEL, V4L2_CID_WHITENESS, V4L2_CID_HCENTER and V4L2_CID_VCENTER were deprecated.
3. A *Camera controls class* was added, with the new controls V4L2_CID_EXPOSURE_AUTO, V4L2_CID_EXPOSURE_ABSOLUTE, V4L2_CID_EXPOSURE_AUTO_PRIORITY, V4L2_CID_PAN_RELATIVE, V4L2_CID_TILT_RELATIVE, V4L2_CID_PAN_RESET, V4L2_CID_TILT_RESET,

V4L2_CID_PAN_ABSOLUTE, V4L2_CID_TILT_ABSOLUTE, V4L2_CID_FOCUS_ABSOLUTE, V4L2_CID_FOCUS_RELATIVE and V4L2_CID_FOCUS_AUTO.

4. The VIDIOC_G_MPEGCOMP and VIDIOC_S_MPEGCOMP ioctls, which were superseded by the *extended controls* interface in Linux 2.6.18, where finally removed from the videodev2.h header file.

V4L2 in Linux 2.6.26

1. The pixel formats V4L2_PIX_FMT_Y16 and V4L2_PIX_FMT_SBGGR16 were added.
2. Added user controls V4L2_CID_CHROMA_AGC and V4L2_CID_COLOR_KILLER.

V4L2 in Linux 2.6.27

1. The *ioctl VIDIOC_S_HW_FREQ_SEEK* ioctl and the V4L2_CAP_HW_FREQ_SEEK capability were added.
2. The pixel formats V4L2_PIX_FMT_YVYU, V4L2_PIX_FMT_PCA501, V4L2_PIX_FMT_PCA505, V4L2_PIX_FMT_PCA508, V4L2_PIX_FMT_PCA561, V4L2_PIX_FMT_SGBRG8, V4L2_PIX_FMT_PAC207 and V4L2_PIX_FMT_PJPG were added.

V4L2 in Linux 2.6.28

1. Added V4L2_MPEG_AUDIO_ENCODING_AAC and V4L2_MPEG_AUDIO_ENCODING_AC3 MPEG audio encodings.
2. Added V4L2_MPEG_VIDEO_ENCODING_MPEG_4_AVC MPEG video encoding.
3. The pixel formats V4L2_PIX_FMT_SGRBG10 and V4L2_PIX_FMT_SGRBG10DPCM8 were added.

V4L2 in Linux 2.6.29

1. The VIDIOC_G_CHIP_IDENT ioctl was renamed to VIDIOC_G_CHIP_IDENT_OLD and VIDIOC_DBG_G_CHIP_IDENT was introduced in its place. The old struct struct v4l2_chip_ident was renamed to struct v4l2_chip_ident_old.
2. The pixel formats V4L2_PIX_FMT_VYUY, V4L2_PIX_FMT_NV16 and V4L2_PIX_FMT_NV61 were added.
3. Added camera controls V4L2_CID_ZOOM_ABSOLUTE, V4L2_CID_ZOOM_RELATIVE, V4L2_CID_ZOOM_CONTINUOUS and V4L2_CID_PRIVACY.

V4L2 in Linux 2.6.30

1. New control flag V4L2_CTRL_FLAG_WRITE_ONLY was added.
2. New control V4L2_CID_COLORFX was added.

V4L2 in Linux 2.6.32

1. In order to be easier to compare a V4L2 API and a kernel version, now V4L2 API is numbered using the Linux Kernel version numeration.
2. Finalized the RDS capture API. See *RDS Interface* for more information.
3. Added new capabilities for modulators and RDS encoders.
4. Add description for libv4l API.
5. Added support for string controls via new type V4L2_CTRL_TYPE_STRING.

6. Added V4L2_CID_BAND_STOP_FILTER documentation.
7. Added FM Modulator (FM TX) Extended Control Class: V4L2_CTRL_CLASS_FM_TX and their Control IDs.
8. Added FM Receiver (FM RX) Extended Control Class: V4L2_CTRL_CLASS_FM_RX and their Control IDs.
9. Added Remote Controller chapter, describing the default Remote Controller mapping for media devices.

V4L2 in Linux 2.6.33

1. Added support for Digital Video timings in order to support HDTV receivers and transmitters.

V4L2 in Linux 2.6.34

1. Added V4L2_CID_IRIS_ABSOLUTE and V4L2_CID_IRIS_RELATIVE controls to the *Camera controls class* .

V4L2 in Linux 2.6.37

1. Remove the vtx (videotext/teletext) API. This API was no longer used and no hardware exists to verify the API. Nor were any userspace applications found that used it. It was originally scheduled for removal in 2.6.35.

V4L2 in Linux 2.6.39

1. The old VIDIOC_*_OLD symbols and V4L1 support were removed.
2. Multi-planar API added. Does not affect the compatibility of current drivers and applications. See *multi-planar API* for details.

V4L2 in Linux 3.1

1. VIDIOC_QUERYCAP now returns a per-subsystem version instead of a per-driver one.

   Standardize an error code for invalid ioctl.

   Added V4L2_CTRL_TYPE_BITMASK.

V4L2 in Linux 3.2

1. V4L2_CTRL_FLAG_VOLATILE was added to signal volatile controls to userspace.
2. Add selection API for extended control over cropping and composing. Does not affect the compatibility of current drivers and applications. See *selection API* for details.

V4L2 in Linux 3.3

1. Added V4L2_CID_ALPHA_COMPONENT control to the *User controls class* .
2. Added the device_caps field to struct v4l2_capabilities and added the new V4L2_CAP_DEVICE_CAPS capability.

V4L2 in Linux 3.4

1. Added *JPEG compression control class* .
2. Extended the DV Timings API: *ioctl VIDIOC_ENUM_DV_TIMINGS, VIDIOC_SUBDEV_ENUM_DV_TIMINGS* , *ioctl VIDIOC_QUERY_DV_TIMINGS* and *ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP* .

V4L2 in Linux 3.5

1. Added integer menus, the new type will be V4L2_CTRL_TYPE_INTEGER_MENU.
2. Added selection API for V4L2 subdev interface: *ioctl VIDIOC_SUBDEV_G_SELECTION, VIDIOC_SUBDEV_S_SELECTION* and *VIDIOC_SUBDEV_S_SELECTION* .
3. Added V4L2_COLORFX_ANTIQUE, V4L2_COLORFX_ART_FREEZE, V4L2_COLORFX_AQUA, V4L2_COLORFX_SILHOUETTE, V4L2_COLORFX_SOLARIZATION, V4L2_COLORFX_VIVID and V4L2_COLORFX_ARBITRARY_CBCR menu items to the V4L2_CID_COLORFX control.
4. Added V4L2_CID_COLORFX_CBCR control.
5. Added camera controls V4L2_CID_AUTO_EXPOSURE_BIAS, V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE, V4L2_CID_IMAGE_STABILIZATION, V4L2_CID_ISO_SENSITIVITY, V4L2_CID_ISO_SENSITIVITY_AUTO, V4L2_CID_EXPOSURE_METERING, V4L2_CID_SCENE_MODE, V4L2_CID_3A_LOCK, V4L2_CID_AUTO_FOCUS_START, V4L2_CID_AUTO_FOCUS_STOP, V4L2_CID_AUTO_FOCUS_STATUS and V4L2_CID_AUTO_FOCUS_RANGE.

V4L2 in Linux 3.6

1. Replaced input in struct *v4l2_buffer* by reserved2 and removed V4L2_BUF_FLAG_INPUT.
2. Added V4L2_CAP_VIDEO_M2M and V4L2_CAP_VIDEO_M2M_MPLANE capabilities.
3. Added support for frequency band enumerations: *ioctl VIDIOC_ENUM_FREQ_BANDS* .

V4L2 in Linux 3.9

1. Added timestamp types to flags field in struct *v4l2_buffer*. See *Buffer Flags* .
2. Added V4L2_EVENT_CTRL_CH_RANGE control event changes flag. See *Control Changes* .

V4L2 in Linux 3.10

1. Removed obsolete and unused DV_PRESET ioctls VIDIOC_G_DV_PRESET, VIDIOC_S_DV_PRESET, VIDIOC_QUERY_DV_PRESET and VIDIOC_ENUM_DV_PRESET. Remove the related v4l2_input/output capability flags V4L2_IN_CAP_PRESETS and V4L2_OUT_CAP_PRESETS.
2. Added new debugging ioctl *ioctl VIDIOC_DBG_G_CHIP_INFO* .

V4L2 in Linux 3.11

1. Remove obsolete VIDIOC_DBG_G_CHIP_IDENT ioctl.

V4L2 in Linux 3.14

1. In struct *v4l2_rect*, the type of width and height fields changed from _s32 to _u32.

V4L2 in Linux 3.15

1. Added Software Defined Radio (SDR) Interface.

V4L2 in Linux 3.16

1. Added event V4L2_EVENT_SOURCE_CHANGE.

V4L2 in Linux 3.17

1. Extended struct *v4l2_pix_format*. Added format flags.
2. Added compound control types and *VIDIOC_QUERY_EXT_CTRL* .

V4L2 in Linux 3.18

1. Added V4L2_CID_PAN_SPEED and V4L2_CID_TILT_SPEED camera controls.

V4L2 in Linux 3.19

1. Rewrote Colorspace chapter, added new enum *v4l2_ycbcr_encoding* and enum *v4l2_quantization* fields to struct *v4l2_pix_format*, struct *v4l2_pix_format_mplane* and struct *v4l2_mbus_framefmt*.

V4L2 in Linux 4.4

1. Renamed V4L2_TUNER_ADC to V4L2_TUNER_SDR. The use of V4L2_TUNER_ADC is deprecated now.
2. Added V4L2_CID_RF_TUNER_RF_GAIN RF Tuner control.
3. Added transmitter support for Software Defined Radio (SDR) Interface.

Relation of V4L2 to other Linux multimedia APIs

X Video Extension

The X Video Extension (abbreviated XVideo or just Xv) is an extension of the X Window system, implemented for example by the XFree86 project. Its scope is similar to V4L2, an API to video capture and output devices for X clients. Xv allows applications to display live video in a window, send window contents to a TV output, and capture or output still images in XPixmaps [1]. With their implementation XFree86 makes the extension available across many operating systems and architectures.

Because the driver is embedded into the X server Xv has a number of advantages over the V4L2 *video overlay interface* . The driver can easily determine the overlay target, i. e. visible graphics memory or off-screen buffers for a destructive overlay. It can program the RAMDAC for a non-destructive overlay, scaling or color-keying, or the clipping functions of the video capture hardware, always in sync with drawing operations or windows moving or changing their stacking order.

To combine the advantages of Xv and V4L a special Xv driver exists in XFree86 and XOrg, just programming any overlay capable Video4Linux device it finds. To enable it /etc/X11/XF86Config must contain these lines:

[1] This is not implemented in XFree86.

```
Section "Module"
    Load "v4l"
EndSection
```

As of XFree86 4.2 this driver still supports only V4L ioctls, however it should work just fine with all V4L2 devices through the V4L2 backward-compatibility layer. Since V4L2 permits multiple opens it is possible (if supported by the V4L2 driver) to capture video while an X client requested video overlay. Restrictions of simultaneous capturing and overlay are discussed in *Video Overlay Interface* apply.

Only marginally related to V4L2, XFree86 extended Xv to support hardware YUV to RGB conversion and scaling for faster video playback, and added an interface to MPEG-2 decoding hardware. This API is useful to display images captured with V4L2 devices.

Digital Video

V4L2 does not support digital terrestrial, cable or satellite broadcast. A separate project aiming at digital receivers exists. You can find its homepage at https://linuxtv.org. The Linux DVB API has no connection to the V4L2 API except that drivers for hybrid hardware may support both.

Audio Interfaces

[to do - OSS/ALSA]

Experimental API Elements

The following V4L2 API elements are currently experimental and may change in the future.

- *ioctl VIDIOC_DBG_G_REGISTER, VIDIOC_DBG_S_REGISTER* and *VIDIOC_DBG_S_REGISTER* ioctls.
- *ioctl VIDIOC_DBG_G_CHIP_INFO* ioctl.

Obsolete API Elements

The following V4L2 API elements were superseded by new interfaces and should not be implemented in new drivers.

- `VIDIOC_G_MPEGCOMP` and `VIDIOC_S_MPEGCOMP` ioctls. Use Extended Controls, *Extended Controls* .
- VIDIOC_G_DV_PRESET, VIDIOC_S_DV_PRESET, VIDIOC_ENUM_DV_PRESETS and VIDIOC_QUERY_DV_PRESET ioctls. Use the DV Timings API (*Digital Video (DV) Timings*).
- `VIDIOC_SUBDEV_G_CROP` and `VIDIOC_SUBDEV_S_CROP` ioctls. Use `VIDIOC_SUBDEV_G_SELECTION` and `VIDIOC_SUBDEV_S_SELECTION`, *ioctl VIDIOC_SUBDEV_G_SELECTION, VIDIOC_SUBDEV_S_SELECTION* .

1.2.8 Function Reference

V4L2 close()

Name

v4l2-close - Close a V4L2 device

Synopsis

```
#include <unistd.h>
```

int **close**(int *fd*)

Arguments

fd File descriptor returned by *open()* .

Description

Closes the device. Any I/O in progress is terminated and resources associated with the file descriptor are freed. However data format parameters, current input or output, control values or other properties remain unchanged.

Return Value

The function returns 0 on success, -1 on failure and the `errno` is set appropriately. Possible error codes:

EBADF `fd` is not a valid open file descriptor.

V4L2 ioctl()

Name

v4l2-ioctl - Program a V4L2 device

Synopsis

```
#include <sys/ioctl.h>
```

int **ioctl**(int *fd*, int *request*, void **argp*)

Arguments

fd File descriptor returned by *open()* .

request V4L2 ioctl request code as defined in the `videodev2.h` header file, for example VIDIOC_QUERYCAP.

argp Pointer to a function parameter, usually a structure.

Description

The *ioctl()* function is used to program V4L2 devices. The argument `fd` must be an open file descriptor. An ioctl `request` has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument `argp` in bytes. Macros and defines specifying V4L2 ioctl requests are located in the `videodev2.h` header file. Applications should use their own copy, not include the version in the kernel sources on the system they compile on. All V4L2 ioctl requests, their respective function and parameters are specified in *Function Reference* .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

When an ioctl that takes an output or read/write parameter fails, the parameter remains unmodified.

ioctl VIDIOC_CREATE_BUFS

Name

VIDIOC_CREATE_BUFS - Create buffers for Memory Mapped or User Pointer or DMA Buffer I/O

Synopsis

int **ioctl**(int *fd*, *VIDIOC_CREATE_BUFS*, struct *v4l2_create_buffers* *argp)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

This ioctl is used to create buffers for *memory mapped* or *user pointer* or *DMA buffer* I/O. It can be used as an alternative or in addition to the *ioctl VIDIOC_REQBUFS* ioctl, when a tighter control over buffers is required. This ioctl can be called multiple times to create buffers of different sizes.

To allocate the device buffers applications must initialize the relevant fields of the struct *v4l2_create_buffers* structure. The `count` field must be set to the number of requested buffers, the `memory` field specifies the requested I/O method and the `reserved` array must be zeroed.

The `format` field specifies the image format that the buffers must be able to handle. The application has to fill in this struct *v4l2_format*. Usually this will be done using the *VIDIOC_TRY_FMT* or *VIDIOC_G_FMT* ioctls to ensure that the requested format is supported by the driver. Based on the format's `type` field the requested buffer size (for single-planar) or plane sizes (for multi-planar formats) will be used for the allocated buffers. The driver may return an error if the size(s) are not supported by the hardware (usually because they are too small).

The buffers created by this ioctl will have as minimum size the size defined by the `format.pix.sizeimage` field (or the corresponding fields for other format types). Usually if the `format.pix.sizeimage` field is less than the minimum required for the given format, then an error will be returned since drivers will typically not allow this. If it is larger, then the value will be used as-is. In other words, the driver may reject the requested size, but if it is accepted the driver will use it unchanged.

When the ioctl is called with a pointer to this structure the driver will attempt to allocate up to the requested number of buffers and store the actual number allocated and the starting index in the `count` and the `index` fields respectively. On return `count` can be smaller than the number requested.

v4l2_create_buffers

Table 1.48: struct v4l2_create_buffers

__u32	index	The starting buffer index, returned by the driver.
__u32	count	The number of buffers requested or granted. If count == 0, then *ioctl VIDIOC_CREATE_BUFS* will set index to the current number of created buffers, and it will check the validity of memory and format.type. If those are invalid -1 is returned and errno is set to EINVAL error code, otherwise *ioctl VIDIOC_CREATE_BUFS* returns 0. It will never set errno to EBUSY error code in this particular case.
__u32	memory	Applications set this field to V4L2_MEMORY_MMAP, V4L2_MEMORY_DMABUF or V4L2_MEMORY_USERPTR. See *v4l2_memory*
struct *v4l2_format*	format	Filled in by the application, preserved by the driver.
__u32	reserved[8]	A place holder for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENOMEM No memory to allocate buffers for *memory mapped* I/O.

EINVAL The buffer type (format.type field), requested I/O method (memory) or format (format field) is not valid.

ioctl VIDIOC_CROPCAP

Name

VIDIOC_CROPCAP - Information about the video cropping and scaling abilities

Synopsis

int **ioctl**(int *fd*, *VIDIOC_CROPCAP*, struct *v4l2_cropcap* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Applications use this function to query the cropping limits, the pixel aspect of images and to calculate scale factors. They set the type field of a v4l2_cropcap structure to the respective buffer (stream) type and call the *ioctl VIDIOC_CROPCAP* ioctl with a pointer to this structure. Drivers fill the rest of the structure. The results are constant except when switching the video standard. Remember this switch can occur implicit when switching the video input or output.

Do not use the multiplanar buffer types. Use V4L2_BUF_TYPE_VIDEO_CAPTURE instead of V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE and use V4L2_BUF_TYPE_VIDEO_OUTPUT instead of V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE.

This ioctl must be implemented for video capture or output devices that support cropping and/or scaling and/or have non-square pixels, and for overlay devices.

v4l2_cropcap

Table 1.49: struct v4l2_cropcap

__u32	type	Type of the data stream, set by the application. Only these types are valid here: V4L2_BUF_TYPE_VIDEO_CAPTURE, V4L2_BUF_TYPE_VIDEO_OUTPUT and V4L2_BUF_TYPE_VIDEO_OVERLAY. See *v4l2_buf_type*.
struct *v4l2_rect*	bounds	Defines the window within capturing or output is possible, this may exclude for example the horizontal and vertical blanking areas. The cropping rectangle cannot exceed these limits. Width and height are defined in pixels, the driver writer is free to choose origin and units of the coordinate system in the analog domain.
struct *v4l2_rect*	defrect	Default cropping rectangle, it shall cover the "whole picture". Assuming pixel aspect 1/1 this could be for example a 640 × 480 rectangle for NTSC, a 768 × 576 rectangle for PAL and SECAM centered over the active picture area. The same co-ordinate system as for bounds is used.
struct *v4l2_fract*	pixelaspect	This is the pixel aspect (y / x) when no scaling is applied, the ratio of the actual sampling frequency and the frequency required to get square pixels. When cropping coordinates refer to square pixels, the driver sets pixelaspect to 1/1. Other common values are 54/59 for PAL and SECAM, 11/10 for NTSC sampled according to [*ITU BT.601*].

Table 1.50: struct v4l2_rect

__s32	left	Horizontal offset of the top, left corner of the rectangle, in pixels.
__s32	top	Vertical offset of the top, left corner of the rectangle, in pixels.
__u32	width	Width of the rectangle, in pixels.
__u32	height	Height of the rectangle, in pixels.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_cropcap* type is invalid.

ENODATA Cropping is not supported for this input or output.

ioctl VIDIOC_DBG_G_CHIP_INFO

Name

VIDIOC_DBG_G_CHIP_INFO - Identify the chips on a TV card

Synopsis

int **ioctl**(int *fd*, *VIDIOC_DBG_G_CHIP_INFO*, struct *v4l2_dbg_chip_info* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Note:

This is an Experimental API Elements *interface and may change in the future.*

For driver debugging purposes this ioctl allows test applications to query the driver about the chips present on the TV card. Regular applications must not use it. When you found a chip specific bug, please contact the linux-media mailing list (https://linuxtv.org/lists.php) so it can be fixed.

Additionally the Linux kernel must be compiled with the CONFIG_VIDEO_ADV_DEBUG option to enable this ioctl.

To query the driver applications must initialize the match.type and match.addr or match.name fields of a struct *v4l2_dbg_chip_info* and call *ioctl VIDIOC_DBG_G_CHIP_INFO* with a pointer to this structure. On success the driver stores information about the selected chip in the name and flags fields.

When match.type is V4L2_CHIP_MATCH_BRIDGE, match.addr selects the nth bridge 'chip' on the TV card. You can enumerate all chips by starting at zero and incrementing match.addr by one until *ioctl VIDIOC_DBG_G_CHIP_INFO* fails with an EINVAL error code. The number zero always selects the bridge chip itself, e. g. the chip connected to the PCI or USB bus. Non-zero numbers identify specific parts of the bridge chip such as an AC97 register block.

When match.type is V4L2_CHIP_MATCH_SUBDEV, match.addr selects the nth sub-device. This allows you to enumerate over all sub-devices.

On success, the name field will contain a chip name and the flags field will contain V4L2_CHIP_FL_READABLE if the driver supports reading registers from the device or V4L2_CHIP_FL_WRITABLE if the driver supports writing registers to the device.

We recommended the v4l2-dbg utility over calling this ioctl directly. It is available from the LinuxTV v4l-dvb repository; see https://linuxtv.org/repo/ for access instructions.

Table 1.51: struct v4l2_dbg_match

__u32	type	See *Chip Match Types* for a list of possible types.	
union	(anonymous)		
	__u32	addr	Match a chip by this number, interpreted according to the type field.
	char	name[32]	Match a chip by this name, interpreted according to the type field. Currently unused.

v4l2_dbg_chip_info

Table 1.52: struct v4l2_dbg_chip_info

struct v4l2_dbg_match	match	How to match the chip, see *struct v4l2_dbg_match* .
char	name[32]	The name of the chip.
__u32	flags	Set by the driver. If V4L2_CHIP_FL_READABLE is set, then the driver supports reading registers from the device. If V4L2_CHIP_FL_WRITABLE is set, then it supports writing registers.
__u32	reserved[8]	Reserved fields, both application and driver must set these to 0.

Table 1.53: Chip Match Types

V4L2_CHIP_MATCH_BRIDGE	0	Match the nth chip on the card, zero for the bridge chip. Does not match sub-devices.
V4L2_CHIP_MATCH_SUBDEV	4	Match the nth sub-device.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The match_type is invalid or no device could be matched.

ioctl VIDIOC_DBG_G_REGISTER, VIDIOC_DBG_S_REGISTER

Name

VIDIOC_DBG_G_REGISTER - VIDIOC_DBG_S_REGISTER - Read or write hardware registers

Synopsis

int **ioctl**(int *fd*, *VIDIOC_DBG_G_REGISTER*, struct *v4l2_dbg_register* *argp)

int **ioctl**(int *fd*, *VIDIOC_DBG_S_REGISTER*, const struct *v4l2_dbg_register* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Note:

This is an Experimental API Elements *interface and may change in the future.*

For driver debugging purposes these ioctls allow test applications to access hardware registers directly. Regular applications must not use them.

Since writing or even reading registers can jeopardize the system security, its stability and damage the hardware, both ioctls require superuser privileges. Additionally the Linux kernel must be compiled with the CONFIG_VIDEO_ADV_DEBUG option to enable these ioctls.

To write a register applications must initialize all fields of a struct *v4l2_dbg_register* except for size and call VIDIOC_DBG_S_REGISTER with a pointer to this structure. The match.type and match.addr or match.name fields select a chip on the TV card, the reg field specifies a register number and the val field the value to be written into the register.

To read a register applications must initialize the match.type, match.addr or match.name and reg fields, and call VIDIOC_DBG_G_REGISTER with a pointer to this structure. On success the driver stores the register value in the val field and the size (in bytes) of the value in size.

When match.type is V4L2_CHIP_MATCH_BRIDGE, match.addr selects the nth non-sub-device chip on the TV card. The number zero always selects the host chip, e. g. the chip connected to the PCI or USB bus. You can find out which chips are present with the *ioctl VIDIOC_DBG_G_CHIP_INFO* ioctl.

When match.type is V4L2_CHIP_MATCH_SUBDEV, match.addr selects the nth sub-device.

These ioctls are optional, not all drivers may support them. However when a driver supports these ioctls it must also support *ioctl VIDIOC_DBG_G_CHIP_INFO* . Conversely it may support VIDIOC_DBG_G_CHIP_INFO but not these ioctls.

VIDIOC_DBG_G_REGISTER and VIDIOC_DBG_S_REGISTER were introduced in Linux 2.6.21, but their API was changed to the one described here in kernel 2.6.29.

We recommended the v4l2-dbg utility over calling these ioctls directly. It is available from the LinuxTV v4l-dvb repository; see https://linuxtv.org/repo/ for access instructions.

v4l2_dbg_match

Table 1.54: struct v4l2_dbg_match

__u32	type	See *Chip Match Types* for a list of possible types.	
union	(anonymous)		
	__u32	addr	Match a chip by this number, interpreted according to the type field.
	char	name[32]	Match a chip by this name, interpreted according to the type field. Currently unused.

v4l2_dbg_register

Table 1.55: struct v4l2_dbg_register

struct v4l2_dbg_match	match	How to match the chip, see *v4l2_dbg_match*.
__u32	size	The register size in bytes.
__u64	reg	A register number.
__u64	val	The value read from, or to be written into the register.

Table 1.56: Chip Match Types

V4L2_CHIP_MATCH_BRIDGE	0	Match the nth chip on the card, zero for the bridge chip. Does not match sub-devices.
V4L2_CHIP_MATCH_SUBDEV	4	Match the nth sub-device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EPERM Insufficient permissions. Root privileges are required to execute these ioctls.

ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD

Name

VIDIOC_DECODER_CMD - VIDIOC_TRY_DECODER_CMD - Execute an decoder command

Synopsis

int **ioctl**(int *fd*, *VIDIOC_DECODER_CMD*, struct *v4l2_decoder_cmd* *argp)

int **ioctl**(int *fd*, *VIDIOC_TRY_DECODER_CMD*, struct *v4l2_decoder_cmd* *argp)

Arguments

fd File descriptor returned by *open()* .

argp pointer to struct *v4l2_decoder_cmd*.

Description

These ioctls control an audio/video (usually MPEG-) decoder. `VIDIOC_DECODER_CMD` sends a command to the decoder, `VIDIOC_TRY_DECODER_CMD` can be used to try a command without actually executing it. To send a command applications must initialize all fields of a struct *v4l2_decoder_cmd* and call `VIDIOC_DECODER_CMD` or `VIDIOC_TRY_DECODER_CMD` with a pointer to this structure.

The `cmd` field must contain the command code. Some commands use the `flags` field for additional information.

A *write()* or *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* call sends an implicit START command to the decoder if it has not been started yet.

A *close()* or *VIDIOC_STREAMOFF* call of a streaming file descriptor sends an implicit immediate STOP command to the decoder, and all buffered data is discarded.

These ioctls are optional, not all drivers may support them. They were introduced in Linux 3.3.

v4l2_decoder_cmd

Table 1.57: struct v4l2_decoder_cmd

__u32	cmd			The decoder command, see *Decoder Commands* .

Continued on next page

Table 1.57 – continued from previous page

__u32	flags			Flags to go with the command. If no flags are defined for this command, drivers and applications must set this field to zero.
union	(anonymous)			
	struct	start		Structure containing additional data for the V4L2_DEC_CMD_START command.
		__s32	speed	Playback speed and direction. The playback speed is defined as speed/1000 of the normal speed. So 1000 is normal playback. Negative numbers denote reverse playback, so -1000 does reverse playback at normal speed. Speeds -1, 0 and 1 have special meanings: speed 0 is shorthand for 1000 (normal playback). A speed of 1 steps just one frame forward, a speed of -1 steps just one frame back.
		__u32	format	Format restrictions. This field is set by the driver, not the application. Possible values are V4L2_DEC_START_FMT_NONE if there are no format restrictions or V4L2_DEC_START_FMT_GOP if the decoder operates on full GOPs (*Group Of Pictures*). This is usually the case for reverse playback: the decoder needs full GOPs, which it can then play in reverse order. So to implement reverse playback the application must feed the decoder the last GOP in the video file, then the GOP before that, etc. etc.
	struct	stop		Structure containing additional data for the V4L2_DEC_CMD_STOP command.
		__u64	pts	Stop playback at this pts or immediately if the playback is already past that timestamp. Leave to 0 if you want to stop after the last frame was decoded.
	struct	raw		
		__u32	data[16]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 1.58: Decoder Commands

V4L2_DEC_CMD_START	0	Start the decoder. When the decoder is already running or paused, this command will just change the playback speed. That means that calling V4L2_DEC_CMD_START when the decoder was paused will *not* resume the decoder. You have to explicitly call V4L2_DEC_CMD_RESUME for that. This command has one flag: V4L2_DEC_CMD_START_MUTE_AUDIO. If set, then audio will be muted when playing back at a non-standard speed.
V4L2_DEC_CMD_STOP	1	Stop the decoder. When the decoder is already stopped, this command does nothing. This command has two flags: if V4L2_DEC_CMD_STOP_TO_BLACK is set, then the decoder will set the picture to black after it stopped decoding. Otherwise the last image will repeat. mem2mem decoders will stop producing new frames altogether. They will send a V4L2_EVENT_EOS event when the last frame has been decoded and all frames are ready to be dequeued and will set the V4L2_BUF_FLAG_LAST buffer flag on the last buffer of the capture queue to indicate there will be no new buffers produced to dequeue. This buffer may be empty, indicated by the driver setting the bytesused field to 0. Once the V4L2_BUF_FLAG_LAST flag was set, the *VIDIOC_DQBUF* ioctl will not block anymore, but return an EPIPE error code. If V4L2_DEC_CMD_STOP_IMMEDIATELY is set, then the decoder stops immediately (ignoring the pts value), otherwise it will keep decoding until timestamp >= pts or until the last of the pending data from its internal buffers was decoded.
V4L2_DEC_CMD_PAUSE	2	Pause the decoder. When the decoder has not been started yet, the driver will return an EPERM error code. When the decoder is already paused, this command does nothing. This command has one flag: if V4L2_DEC_CMD_PAUSE_TO_BLACK is set, then set the decoder output to black when paused.
V4L2_DEC_CMD_RESUME	3	Resume decoding after a PAUSE command. When the decoder has not been started yet, the driver will return an EPERM error code. When the decoder is already running, this command does nothing. No flags are defined for this command.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The cmd field is invalid.

EPERM The application sent a PAUSE or RESUME command when the decoder was not running.

ioctl VIDIOC_DQEVENT

Name

VIDIOC_DQEVENT - Dequeue event

Synopsis

int **ioctl**(int *fd*, *VIDIOC_DQEVENT*, struct *v4l2_event* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Dequeue an event from a video device. No input is required for this ioctl. All the fields of the struct *v4l2_event* structure are filled by the driver. The file handle will also receive exceptions which the application may get by e.g. using the select system call.

v4l2_event

Table 1.59: struct v4l2_event

__u32	type		Type of the event, see *Event Types* .
union	u		
	struct *v4l2_event_vsync*	vsync	Event data for event V4L2_EVENT_VSYNC.
	struct *v4l2_event_ctrl*	ctrl	Event data for event V4L2_EVENT_CTRL.
	struct *v4l2_event_frame_sync*	frame_sync	Event data for event V4L2_EVENT_FRAME_SYNC.
	struct *v4l2_event_motion_det*	motion_det	Event data for event V4L2_EVENT_MOTION_DET.
	struct *v4l2_event_src_change*	src_change	Event data for event V4L2_EVENT_SOURCE_CHANGE.
	__u8	data[64]	Event data. Defined by the event type. The union should be used to define easily accessible type for events.
__u32	pending		Number of pending events excluding this one.
__u32	sequence		Event sequence number. The sequence number is incremented for every subscribed event that takes place. If sequence numbers are not contiguous it means that events have been lost.
struct timespec	timestamp		Event timestamp. The timestamp has been taken from the CLOCK_MONOTONIC clock. To access the same clock outside V4L2, use clock_gettime().
u32	id		The ID associated with the event source. If the event does not have an associated ID (this depends on the event type), then this is 0.
__u32	reserved[8]		Reserved for future extensions. Drivers must set the array to zero.

Table 1.60: Event Types

V4L2_EVENT_ALL	0	All events. V4L2_EVENT_ALL is valid only for VIDIOC_UNSUBSCRIBE_EVENT for unsubscribing all events at once.
V4L2_EVENT_VSYNC	1	This event is triggered on the vertical sync. This event has a struct *v4l2_event_vsync* associated with it.

Continued on next page

Table 1.60 – continued from previous page

V4L2_EVENT_EOS	2	This event is triggered when the end of a stream is reached. This is typically used with MPEG decoders to report to the application when the last of the MPEG stream has been decoded.
V4L2_EVENT_CTRL	3	This event requires that the id matches the control ID from which you want to receive events. This event is triggered if the control's value changes, if a button control is pressed or if the control's flags change. This event has a struct *v4l2_event_ctrl* associated with it. This struct contains much of the same information as struct *v4l2_queryctrl* and struct *v4l2_control*. If the event is generated due to a call to *VIDIOC_S_CTRL* or *VIDIOC_S_EXT_CTRLS* , then the event will *not* be sent to the file handle that called the ioctl function. This prevents nasty feedback loops. If you *do* want to get the event, then set the V4L2_EVENT_SUB_FL_ALLOW_FEEDBACK flag. This event type will ensure that no information is lost when more events are raised than there is room internally. In that case the struct *v4l2_event_ctrl* of the second-oldest event is kept, but the changes field of the second-oldest event is ORed with the changes field of the oldest event.
V4L2_EVENT_FRAME_SYNC	4	Triggered immediately when the reception of a frame has begun. This event has a struct *v4l2_event_frame_sync* associated with it. If the hardware needs to be stopped in the case of a buffer underrun it might not be able to generate this event. In such cases the frame_sequence field in struct *v4l2_event_frame_sync* will not be incremented. This causes two consecutive frame sequence numbers to have n times frame interval in between them.
V4L2_EVENT_SOURCE_CHANGE	5	This event is triggered when a source parameter change is detected during runtime by the video device. It can be a runtime resolution change triggered by a video decoder or the format change happening on an input connector. This event requires that the id matches the input index (when used with a video device node) or the pad index (when used with a subdevice node) from which you want to receive events. This event has a struct *v4l2_event_src_change* associated with it. The changes bitfield denotes what has changed for the subscribed pad. If multiple events occurred before application could dequeue them, then the changes will have the ORed value of all the events generated.
V4L2_EVENT_MOTION_DET	6	Triggered whenever the motion detection state for one or more of the regions changes. This event has a struct *v4l2_event_motion_det* associated with it.
V4L2_EVENT_PRIVATE_START	0x08000000	Base event number for driver-private events.

v4l2_event_vsync

Table 1.61: struct v4l2_event_vsync

__u8	field	The upcoming field. See enum *v4l2_field*.

v4l2_event_ctrl

Table 1.62: struct v4l2_event_ctrl

__u32	changes		A bitmask that tells what has changed. See *Control Changes* .
__u32	type		The type of the control. See enum *v4l2_ctrl_type*.
union (anonymous)			
	__s32	value	The 32-bit value of the control for 32-bit control types. This is 0 for string controls since the value of a string cannot be passed using *ioctl VIDIOC_DQEVENT* .
	__s64	value64	The 64-bit value of the control for 64-bit control types.
__u32	flags		The control flags. See *Control Flags* .
__s32	minimum		The minimum value of the control. See struct *v4l2_queryctrl* .
__s32	maximum		The maximum value of the control. See struct *v4l2_queryctrl* .
__s32	step		The step value of the control. See struct *v4l2_queryctrl* .
__s32	default_value		The default value value of the control. See struct *v4l2_queryctrl* .

v4l2_event_frame_sync

Table 1.63: struct v4l2_event_frame_sync

__u32	frame_sequence	The sequence number of the frame being received.

v4l2_event_src_change

Table 1.64: struct v4l2_event_src_change

__u32	changes	A bitmask that tells what has changed. See *Source Changes* .

v4l2_event_motion_det

Table 1.65: struct v4l2_event_motion_det

__u32	flags	Currently only one flag is available: if V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ is set, then the frame_sequence field is valid, otherwise that field should be ignored.
__u32	frame_sequence	The sequence number of the frame being received. Only valid if the V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ flag was set.
__u32	region_mask	The bitmask of the regions that reported motion. There is at least one region. If this field is 0, then no motion was detected at all. If there is no V4L2_CID_DETECT_MD_REGION_GRID control (see *Detect Control Reference*) to assign a different region to each cell in the motion detection grid, then that all cells are automatically assigned to the default region 0.

Table 1.66: Control Changes

V4L2_EVENT_CTRL_CH_VALUE	0x0001	This control event was triggered because the value of the control changed. Special cases: Volatile controls do no generate this event; If a control has the V4L2_CTRL_FLAG_EXECUTE_ON_WRITE flag set, then this event is sent as well, regardless its value.
V4L2_EVENT_CTRL_CH_FLAGS	0x0002	This control event was triggered because the control flags changed.
V4L2_EVENT_CTRL_CH_RANGE	0x0004	This control event was triggered because the minimum, maximum, step or the default value of the control changed.

Table 1.67: Source Changes

V4L2_EVENT_SRC_CH_RESOLUTION	0x0001	This event gets triggered when a resolution change is detected at an input. This can come from an input connector or from a video decoder.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP

Name

VIDIOC_DV_TIMINGS_CAP - VIDIOC_SUBDEV_DV_TIMINGS_CAP - The capabilities of the Digital Video receiver/transmitter

Synopsis

int **ioctl**(int *fd*, *VIDIOC_DV_TIMINGS_CAP*, struct *v4l2_dv_timings_cap* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_DV_TIMINGS_CAP*, struct *v4l2_dv_timings_cap* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the capabilities of the DV receiver/transmitter applications initialize the pad field to 0, zero the reserved array of struct *v4l2_dv_timings_cap* and call the VIDIOC_DV_TIMINGS_CAP ioctl on a video node and the driver will fill in the structure.

Note:

Drivers may return different values after switching the video input or output.

When implemented by the driver DV capabilities of subdevices can be queried by calling the VIDIOC_SUBDEV_DV_TIMINGS_CAP ioctl directly on a subdevice node. The capabilities are specific to inputs (for DV receivers) or outputs (for DV transmitters), applications must specify the desired pad number in the struct *v4l2_dv_timings_cap* pad field and zero the reserved array. Attempts to query capabilities on a pad that doesn't support them will return an EINVAL error code.

v4l2_bt_timings_cap

Table 1.68: struct v4l2_bt_timings_cap

__u32	min_width	Minimum width of the active video in pixels.
__u32	max_width	Maximum width of the active video in pixels.
__u32	min_height	Minimum height of the active video in lines.
__u32	max_height	Maximum height of the active video in lines.
__u64	min_pixelclock	Minimum pixelclock frequency in Hz.
__u64	max_pixelclock	Maximum pixelclock frequency in Hz.
__u32	standards	The video standard(s) supported by the hardware. See *DV BT Timing standards* for a list of standards.
__u32	capabilities	Several flags giving more information about the capabilities. See *DV BT Timing capabilities* for a description of the flags.
__u32	reserved[16]	Reserved for future extensions. Drivers must set the array to zero.

v4l2_dv_timings_cap

Table 1.69: struct v4l2_dv_timings_cap

__u32	type	Type of DV timings as listed in *DV Timing types* .	
__u32	pad	Pad number as reported by the media controller API. This field is only used when operating on a subdevice node. When operating on a video node applications must set this field to zero.	
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.	
union			
	struct *v4l2_bt_timings_cap*	bt	BT.656/1120 timings capabilities of the hardware.
	__u32	raw_data[32]	

Table 1.70: DV BT Timing capabilities

Flag	Description
V4L2_DV_BT_CAP_INTERLACED	Interlaced formats are supported.
V4L2_DV_BT_CAP_PROGRESSIVE	Progressive formats are supported.
V4L2_DV_BT_CAP_REDUCED_BLANKING	CVT/GTF specific: the timings can make use of reduced blanking (CVT) or the 'Secondary GTF' curve (GTF).
V4L2_DV_BT_CAP_CUSTOM	Can support non-standard timings, i.e. timings not belonging to the standards set in the standards field.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_ENCODER_CMD, VIDIOC_TRY_ENCODER_CMD

Name

VIDIOC_ENCODER_CMD - VIDIOC_TRY_ENCODER_CMD - Execute an encoder command

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENCODER_CMD*, struct *v4l2_encoder_cmd* *argp)

int **ioctl**(int *fd*, *VIDIOC_TRY_ENCODER_CMD*, struct *v4l2_encoder_cmd* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

These ioctls control an audio/video (usually MPEG-) encoder. VIDIOC_ENCODER_CMD sends a command to the encoder, VIDIOC_TRY_ENCODER_CMD can be used to try a command without actually executing it.

To send a command applications must initialize all fields of a struct *v4l2_encoder_cmd* and call VIDIOC_ENCODER_CMD or VIDIOC_TRY_ENCODER_CMD with a pointer to this structure.

The cmd field must contain the command code. The flags field is currently only used by the STOP command and contains one bit: If the V4L2_ENC_CMD_STOP_AT_GOP_END flag is set, encoding will continue until the end of the current *Group Of Pictures*, otherwise it will stop immediately.

A *read()* or *VIDIOC_STREAMON* call sends an implicit START command to the encoder if it has not been started yet. After a STOP command, *read()* calls will read the remaining data buffered by the driver. When the buffer is empty, *read()* will return zero and the next *read()* call will restart the encoder.

A *close()* or *VIDIOC_STREAMOFF* call of a streaming file descriptor sends an implicit immediate STOP to the encoder, and all buffered data is discarded.

These ioctls are optional, not all drivers may support them. They were introduced in Linux 2.6.21.

v4l2_encoder_cmd

Table 1.71: struct v4l2_encoder_cmd

__u32	cmd	The encoder command, see *Encoder Commands* .
__u32	flags	Flags to go with the command, see *Encoder Command Flags* . If no flags are defined for this command, drivers and applications must set this field to zero.
__u32	data[8]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 1.72: Encoder Commands

V4L2_ENC_CMD_START	0	Start the encoder. When the encoder is already running or paused, this command does nothing. No flags are defined for this command.
V4L2_ENC_CMD_STOP	1	Stop the encoder. When the V4L2_ENC_CMD_STOP_AT_GOP_END flag is set, encoding will continue until the end of the current *Group Of Pictures*, otherwise encoding will stop immediately. When the encoder is already stopped, this command does nothing. mem2mem encoders will send a V4L2_EVENT_EOS event when the last frame has been encoded and all frames are ready to be dequeued and will set the V4L2_BUF_FLAG_LAST buffer flag on the last buffer of the capture queue to indicate there will be no new buffers produced to dequeue. This buffer may be empty, indicated by the driver setting the bytesused field to 0. Once the V4L2_BUF_FLAG_LAST flag was set, the *VIDIOC_DQBUF* ioctl will not block anymore, but return an EPIPE error code.
V4L2_ENC_CMD_PAUSE	2	Pause the encoder. When the encoder has not been started yet, the driver will return an EPERM error code. When the encoder is already paused, this command does nothing. No flags are defined for this command.
V4L2_ENC_CMD_RESUME	3	Resume encoding after a PAUSE command. When the encoder has not been started yet, the driver will return an EPERM error code. When the encoder is already running, this command does nothing. No flags are defined for this command.

Table 1.73: Encoder Command Flags

V4L2_ENC_CMD_STOP_AT_GOP_END	0x0001	Stop encoding at the end of the current *Group Of Pictures*, rather than immediately.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The `cmd` field is invalid.

EPERM The application sent a PAUSE or RESUME command when the encoder was not running.

ioctl VIDIOC_ENUMAUDIO

Name

VIDIOC_ENUMAUDIO - Enumerate audio inputs

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUMAUDIO*, struct *v4l2_audio* *argp)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

To query the attributes of an audio input applications initialize the `index` field and zero out the `reserved` array of a struct *v4l2_audio* and call the *ioctl VIDIOC_ENUMAUDIO* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all audio inputs applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

See *VIDIOC_G_AUDIO* for a description of struct *v4l2_audio*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The number of the audio input is out of bounds.

ioctl VIDIOC_ENUMAUDOUT

Name

VIDIOC_ENUMAUDOUT - Enumerate audio outputs

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUMAUDOUT*, struct *v4l2_audioout* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the attributes of an audio output applications initialize the index field and zero out the reserved array of a struct *v4l2_audioout* and call the VIDIOC_G_AUDOUT ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code when the index is out of bounds. To enumerate all audio outputs applications shall begin at index zero, incrementing by one until the driver returns EINVAL.

Note:

Connectors on a TV card to loop back the received audio signal to a sound card are not audio outputs in this sense.

See *VIDIOC_G_AUDIOout* for a description of struct *v4l2_audioout*.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The number of the audio output is out of bounds.

ioctl VIDIOC_ENUM_DV_TIMINGS, VIDIOC_SUBDEV_ENUM_DV_TIMINGS

Name

VIDIOC_ENUM_DV_TIMINGS - VIDIOC_SUBDEV_ENUM_DV_TIMINGS - Enumerate supported Digital Video timings

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUM_DV_TIMINGS*, struct *v4l2_enum_dv_timings* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_ENUM_DV_TIMINGS*, struct *v4l2_enum_dv_timings* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

While some DV receivers or transmitters support a wide range of timings, others support only a limited number of timings. With this ioctl applications can enumerate a list of known supported timings. Call *ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP* to check if it also supports other standards or even custom timings that are not in this list.

To query the available timings, applications initialize the `index` field, set the pad field to 0, zero the reserved array of struct *v4l2_enum_dv_timings* and call the `VIDIOC_ENUM_DV_TIMINGS` ioctl on a video node with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all supported DV timings, applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

Note:

Drivers may enumerate a different set of DV timings after switching the video input or output.

When implemented by the driver DV timings of subdevices can be queried by calling the `VIDIOC_SUBDEV_ENUM_DV_TIMINGS` ioctl directly on a subdevice node. The DV timings are specific to inputs (for DV receivers) or outputs (for DV transmitters), applications must specify the desired pad number in the struct *v4l2_enum_dv_timings* pad field. Attempts to enumerate timings on a pad that doesn't support them will return an `EINVAL` error code.

v4l2_enum_dv_timings

Table 1.74: struct v4l2_enum_dv_timings

__u32	`index`	Number of the DV timings, set by the application.
__u32	`pad`	Pad number as reported by the media controller API. This field is only used when operating on a subdevice node. When operating on a video node applications must set this field to zero.
__u32	`reserved[2]`	Reserved for future extensions. Drivers and applications must set the array to zero.
struct *v4l2_dv_timings*	`timings`	The timings.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_enum_dv_timings* `index` is out of bounds or the pad number is invalid.

ENODATA Digital video presets are not supported for this input or output.

ioctl VIDIOC_ENUM_FMT

Name

VIDIOC_ENUM_FMT - Enumerate image formats

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUM_FMT*, struct *v4l2_fmtdesc* **argp*)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To enumerate image formats applications initialize the type and index field of struct *v4l2_fmtdesc* and call the *ioctl VIDIOC_ENUM_FMT* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code. All formats are enumerable by beginning at index zero and incrementing by one until EINVAL is returned.

Note:

After switching input or output the list of enumerated image formats may be different.

v4l2_fmtdesc

Table 1.75: struct v4l2_fmtdesc

__u32	index	Number of the format in the enumeration, set by the application. This is in no way related to the pixelformat field.
__u32	type	Type of the data stream, set by the application. Only these types are valid here: V4L2_BUF_TYPE_VIDEO_CAPTURE, V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE, V4L2_BUF_TYPE_VIDEO_OUTPUT, V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE and V4L2_BUF_TYPE_VIDEO_OVERLAY. See *v4l2_buf_type*.
__u32	flags	See *Image Format Description Flags*
__u8	description[32]	Description of the format, a NUL-terminated ASCII string. This information is intended for the user, for example: “YUV 4:2:2”.
__u32	pixelformat	The image format identifier. This is a four character code as computed by the v4l2_fourcc() macro:

```
#define v4l2_fourcc(a,b,c,d)

(((__u32)(a)<<0)|((__u32)(b)<<8)|((__u32)(c)<<16)|((__u32)(d)<<24))
```

Several image formats are already defined by this specification in *Image Formats* .

Attention:

These codes are not the same as those used in the Windows world.

__u32	reserved[4]	Reserved for future extensions. Drivers must set the array to zero.

Table 1.76: Image Format Description Flags

V4L2_FMT_FLAG_COMPRESSED	0x0001	This is a compressed format.
V4L2_FMT_FLAG_EMULATED	0x0002	This format is not native to the device but emulated through software (usually libv4l2), where possible try to use a native format instead for better performance.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_fmtdesc* `type` is not supported or the `index` is out of bounds.

ioctl VIDIOC_ENUM_FRAMESIZES

Name

VIDIOC_ENUM_FRAMESIZES - Enumerate frame sizes

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUM_FRAMESIZES*, struct *v4l2_frmsizeenum* *argp)

Arguments

fd File descriptor returned by *open()* .

argp Pointer to a struct *v4l2_frmsizeenum* that contains an index and pixel format and receives a frame width and height.

Description

This ioctl allows applications to enumerate all frame sizes (i. e. width and height in pixels) that the device supports for the given pixel format.

The supported pixel formats can be obtained by using the *ioctl VIDIOC_ENUM_FMT* function.

The return value and the content of the `v4l2_frmsizeenum.type` field depend on the type of frame sizes the device supports. Here are the semantics of the function for the different cases:

- **Discrete:** The function returns success if the given index value (zero-based) is valid. The application should increase the index by one for each call until `EINVAL` is returned. The `v4l2_frmsizeenum.type` field is set to `V4L2_FRMSIZE_TYPE_DISCRETE` by the driver. Of the union only the `discrete` member is valid.
- **Step-wise:** The function returns success if the given index value is zero and `EINVAL` for any other index value. The `v4l2_frmsizeenum.type` field is set to `V4L2_FRMSIZE_TYPE_STEPWISE` by the driver. Of the union only the `stepwise` member is valid.
- **Continuous:** This is a special case of the step-wise type above. The function returns success if the given index value is zero and `EINVAL` for any other index value. The `v4l2_frmsizeenum.type` field is set to `V4L2_FRMSIZE_TYPE_CONTINUOUS` by the driver. Of the union only the `stepwise` member is valid and the `step_width` and `step_height` values are set to 1.

When the application calls the function with index zero, it must check the type field to determine the type of frame size enumeration the device supports. Only for the V4L2_FRMSIZE_TYPE_DISCRETE type does it make sense to increase the index value to receive more frame sizes.

Note:

The order in which the frame sizes are returned has no special meaning. In particular does it not say anything about potential default format sizes.

Applications can assume that the enumeration data does not change without any interaction from the application itself. This means that the enumeration data is consistent if the application does not perform any other ioctl calls while it runs the frame size enumeration.

Structs

In the structs below, *IN* denotes a value that has to be filled in by the application, *OUT* denotes values that the driver fills in. The application should zero out all members except for the *IN* fields.

v4l2_frmsize_discrete

Table 1.77: struct v4l2_frmsize_discrete

__u32	width	Width of the frame [pixel].
__u32	height	Height of the frame [pixel].

v4l2_frmsize_stepwise

Table 1.78: struct v4l2_frmsize_stepwise

__u32	min_width	Minimum frame width [pixel].
__u32	max_width	Maximum frame width [pixel].
__u32	step_width	Frame width step size [pixel].
__u32	min_height	Minimum frame height [pixel].
__u32	max_height	Maximum frame height [pixel].
__u32	step_height	Frame height step size [pixel].

v4l2_frmsizeenum

Table 1.79: struct v4l2_frmsizeenum

__u32	index		IN: Index of the given frame size in the enumeration.
__u32	pixel_format		IN: Pixel format for which the frame sizes are enumerated.
__u32	type		OUT: Frame size type the device supports.
union			OUT: Frame size with the given index.
	struct *v4l2_frmsize_discrete*	discrete	
	struct *v4l2_frmsize_stepwise*	stepwise	
__u32	reserved[2]		Reserved space for future use. Must be zeroed by drivers and applications.

Enums

v4l2_frmsizetypes

Table 1.80: enum v4l2_frmsizetypes

V4L2_FRMSIZE_TYPE_DISCRETE	1	Discrete frame size.
V4L2_FRMSIZE_TYPE_CONTINUOUS	2	Continuous frame size.
V4L2_FRMSIZE_TYPE_STEPWISE	3	Step-wise defined frame size.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_ENUM_FRAMEINTERVALS

Name

VIDIOC_ENUM_FRAMEINTERVALS - Enumerate frame intervals

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUM_FRAMEINTERVALS*, struct *v4l2_frmivalenum* *argp)

Arguments

fd File descriptor returned by *open()* .

argp Pointer to a struct *v4l2_frmivalenum* structure that contains a pixel format and size and receives a frame interval.

Description

This ioctl allows applications to enumerate all frame intervals that the device supports for the given pixel format and frame size.

The supported pixel formats and frame sizes can be obtained by using the *ioctl VIDIOC_ENUM_FMT* and *ioctl VIDIOC_ENUM_FRAMESIZES* functions.

The return value and the content of the `v4l2_frmivalenum.type` field depend on the type of frame intervals the device supports. Here are the semantics of the function for the different cases:

- **Discrete:** The function returns success if the given index value (zero-based) is valid. The application should increase the index by one for each call until `EINVAL` is returned. The *v4l2_frmivalenum.type* field is set to *V4L2_FRMIVAL_TYPE_DISCRETE* by the driver. Of the union only the *discrete* member is valid.
- **Step-wise:** The function returns success if the given index value is zero and `EINVAL` for any other index value. The `v4l2_frmivalenum.type` field is set to `V4L2_FRMIVAL_TYPE_STEPWISE` by the driver. Of the union only the `stepwise` member is valid.
- **Continuous:** This is a special case of the step-wise type above. The function returns success if the given index value is zero and `EINVAL` for any other index value. The `v4l2_frmivalenum.type` field is set to `V4L2_FRMIVAL_TYPE_CONTINUOUS` by the driver. Of the union only the `stepwise` member is valid and the `step` value is set to 1.

When the application calls the function with index zero, it must check the `type` field to determine the type of frame interval enumeration the device supports. Only for the `V4L2_FRMIVAL_TYPE_DISCRETE` type does it make sense to increase the index value to receive more frame intervals.

Note:

The order in which the frame intervals are returned has no special meaning. In particular does it not say anything about potential default frame intervals.

Applications can assume that the enumeration data does not change without any interaction from the application itself. This means that the enumeration data is consistent if the application does not perform any other ioctl calls while it runs the frame interval enumeration.

Note:

Frame intervals and frame rates: *The V4L2 API uses frame intervals instead of frame rates. Given the frame interval the frame rate can be computed as follows:*

```
frame_rate = 1 / frame_interval
```

Structs

In the structs below, *IN* denotes a value that has to be filled in by the application, *OUT* denotes values that the driver fills in. The application should zero out all members except for the *IN* fields.

v4l2_frmival_stepwise

Table 1.81: struct v4l2_frmival_stepwise

struct *v4l2_fract*	min	Minimum frame interval [s].
struct *v4l2_fract*	max	Maximum frame interval [s].
struct *v4l2_fract*	step	Frame interval step size [s].

v4l2_frmivalenum

Table 1.82: struct v4l2_frmivalenum

__u32	index		IN: Index of the given frame interval in the enumeration.
__u32	pixel_format		IN: Pixel format for which the frame intervals are enumerated.
__u32	width		IN: Frame width for which the frame intervals are enumerated.
__u32	height		IN: Frame height for which the frame intervals are enumerated.
__u32	type		OUT: Frame interval type the device supports.
union			OUT: Frame interval with the given index.
	struct *v4l2_fract*	discrete	Frame interval [s].
	struct *v4l2_frmival_stepwise*	stepwise	
__u32	reserved[2]		Reserved space for future use. Must be zeroed by drivers and applications.

Enums

v4l2_frmivaltypes

Table 1.83: enum v4l2_frmivaltypes

V4L2_FRMIVAL_TYPE_DISCRETE	1	Discrete frame interval.
V4L2_FRMIVAL_TYPE_CONTINUOUS	2	Continuous frame interval.
V4L2_FRMIVAL_TYPE_STEPWISE	3	Step-wise defined frame interval.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_ENUM_FREQ_BANDS

Name

VIDIOC_ENUM_FREQ_BANDS - Enumerate supported frequency bands

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUM_FREQ_BANDS*, struct *v4l2_frequency_band* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Enumerates the frequency bands that a tuner or modulator supports. To do this applications initialize the tuner, type and index fields, and zero out the reserved array of a struct *v4l2_frequency_band* and call the *ioctl VIDIOC_ENUM_FREQ_BANDS* ioctl with a pointer to this structure.

This ioctl is supported if the V4L2_TUNER_CAP_FREQ_BANDS capability of the corresponding tuner/modulator is set.

v4l2_frequency_band

Table 1.84: struct v4l2_frequency_band

__u32	tuner	The tuner or modulator index number. This is the same value as in the struct *v4l2_input* tuner field and the struct *v4l2_tuner* index field, or the struct *v4l2_output* modulator field and the struct *v4l2_modulator* index field.
__u32	type	The tuner type. This is the same value as in the struct *v4l2_tuner* type field. The type must be set to V4L2_TUNER_RADIO for /dev/radioX device nodes, and to V4L2_TUNER_ANALOG_TV for all others. Set this field to V4L2_TUNER_RADIO for modulators (currently only radio modulators are supported). See *v4l2_tuner_type*
__u32	index	Identifies the frequency band, set by the application.
__u32	capability	The tuner/modulator capability flags for this frequency band, see *Tuner and Modulator Capability Flags* . The V4L2_TUNER_CAP_LOW or V4L2_TUNER_CAP_1HZ capability must be the same for all frequency bands of the selected tuner/modulator. So either all bands have that capability set, or none of them have that capability.
__u32	rangelow	The lowest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, for this frequency band. A 1 Hz unit is used when the capability flag V4L2_TUNER_CAP_1HZ is set.
__u32	rangehigh	The highest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, for this frequency band. A 1 Hz unit is used when the capability flag V4L2_TUNER_CAP_1HZ is set.
__u32	modulation	The supported modulation systems of this frequency band. See *Band Modulation Systems* . ***Note:*** *Currently only one modulation system per frequency band is supported. More work will need to be done if multiple modulation systems are possible. Contact the linux-media mailing list (https://linuxtv.org/lists.php) if you need such functionality.*
__u32	reserved[9]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 1.85: Band Modulation Systems

V4L2_BAND_MODULATION_VSB	0x02	Vestigial Sideband modulation, used for analog TV.
V4L2_BAND_MODULATION_FM	0x04	Frequency Modulation, commonly used for analog radio.
V4L2_BAND_MODULATION_AM	0x08	Amplitude Modulation, commonly used for analog radio.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The `tuner` or `index` is out of bounds or the `type` field is wrong.

ioctl VIDIOC_ENUMINPUT

Name

VIDIOC_ENUMINPUT - Enumerate video inputs

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUMINPUT*, struct *v4l2_input* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the attributes of a video input applications initialize the `index` field of struct *v4l2_input* and call the *ioctl VIDIOC_ENUMINPUT* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all inputs applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

v4l2_input

Table 1.86: struct v4l2_input

__u32	index	Identifies the input, set by the application.
__u8	name[32]	Name of the video input, a NUL-terminated ASCII string, for example: "Vin (Composite 2)". This information is intended for the user, preferably the connector label on the device itself.
__u32	type	Type of the input, see *Input Types* .
__u32	audioset	Drivers can enumerate up to 32 video and audio inputs. This field shows which audio inputs were selectable as audio source if this was the currently selected video input. It is a bit mask. The LSB corresponds to audio input 0, the MSB to input 31. Any number of bits can be set, or none. When the driver does not enumerate audio inputs no bits must be set. Applications shall not interpret this as lack of audio support. Some drivers automatically select audio sources and do not enumerate them since there is no choice anyway. For details on audio inputs and how to select the current input see *Audio Inputs and Outputs* .
__u32	tuner	Capture devices can have zero or more tuners (RF demodulators). When the type is set to V4L2_INPUT_TYPE_TUNER this is an RF connector and this field identifies the tuner. It corresponds to struct *v4l2_tuner* field index. For details on tuners see *Tuners and Modulators* .
v4l2_std_id	std	Every video input supports one or more different video standards. This field is a set of all supported standards. For details on video standards and how to switch see *Video Standards* .
__u32	status	This field provides status information about the input. See *Input Status Flags* for flags. With the exception of the sensor orientation bits status is only valid when this is the current input.
__u32	capabilities	This field provides capabilities for the input. See *Input capabilities* for flags.
__u32	reserved[3]	Reserved for future extensions. Drivers must set the array to zero.

Table 1.87: Input Types

V4L2_INPUT_TYPE_TUNER	1	This input uses a tuner (RF demodulator).
V4L2_INPUT_TYPE_CAMERA	2	Analog baseband input, for example CVBS / Composite Video, S-Video, RGB.
V4L2_INPUT_TYPE_TOUCH	3	This input is a touch device for capturing raw touch data.

Table 1.88: Input Status Flags

General		
V4L2_IN_ST_NO_POWER	0x00000001	Attached device is off.
V4L2_IN_ST_NO_SIGNAL	0x00000002	
V4L2_IN_ST_NO_COLOR	0x00000004	The hardware supports color decoding, but does not detect color modulation in the signal.
Sensor Orientation		
V4L2_IN_ST_HFLIP	0x00000010	The input is connected to a device that produces a signal that is flipped horizontally and does not correct this before passing the signal to userspace.
V4L2_IN_ST_VFLIP	0x00000020	The input is connected to a device that produces a signal that is flipped vertically and does not correct this before passing the signal to userspace. .. note:: A 180 degree rotation is the same as HFLIP | VFLIP
Analog Video		
V4L2_IN_ST_NO_H_LOCK	0x00000100	No horizontal sync lock.
V4L2_IN_ST_COLOR_KILL	0x00000200	A color killer circuit automatically disables color decoding when it detects no color modulation. When this flag is set the color killer is enabled *and* has shut off color decoding.
V4L2_IN_ST_NO_V_LOCK	0x00000400	No vertical sync lock.
V4L2_IN_ST_NO_STD_LOCK	0x00000800	No standard format lock in case of auto-detection format by the component.
Digital Video		
V4L2_IN_ST_NO_SYNC	0x00010000	No synchronization lock.
V4L2_IN_ST_NO_EQU	0x00020000	No equalizer lock.
V4L2_IN_ST_NO_CARRIER	0x00040000	Carrier recovery failed.
VCR and Set-Top Box		
V4L2_IN_ST_MACROVISION	0x01000000	Macrovision is an analog copy prevention system mangling the video signal to confuse video recorders. When this flag is set Macrovision has been detected.
V4L2_IN_ST_NO_ACCESS	0x02000000	Conditional access denied.
V4L2_IN_ST_VTR	0x04000000	VTR time constant. [?]

Table 1.89: Input capabilities

V4L2_IN_CAP_DV_TIMINGS	0x00000002	This input supports setting video timings by using VIDIOC_S_DV_TIMINGS.
V4L2_IN_CAP_STD	0x00000004	This input supports setting the TV standard by using VIDIOC_S_STD.
V4L2_IN_CAP_NATIVE_SIZE	0x00000008	This input supports setting the native size using the V4L2_SEL_TGT_NATIVE_SIZE selection target, see *Common selection definitions* .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_input* index is out of bounds.

ioctl VIDIOC_ENUMOUTPUT

Name

VIDIOC_ENUMOUTPUT - Enumerate video outputs

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUMOUTPUT*, struct *v4l2_output* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the attributes of a video outputs applications initialize the index field of struct *v4l2_output* and call the *ioctl VIDIOC_ENUMOUTPUT* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code when the index is out of bounds. To enumerate all outputs applications shall begin at index zero, incrementing by one until the driver returns EINVAL.

v4l2_output

Table 1.90: struct v4l2_output

__u32	index	Identifies the output, set by the application.
__u8	name[32]	Name of the video output, a NUL-terminated ASCII string, for example: “Vout”. This information is intended for the user, preferably the connector label on the device itself.
__u32	type	Type of the output, see *Output Type* .
__u32	audioset	Drivers can enumerate up to 32 video and audio outputs. This field shows which audio outputs were selectable as the current output if this was the currently selected video output. It is a bit mask. The LSB corresponds to audio output 0, the MSB to output 31. Any number of bits can be set, or none. When the driver does not enumerate audio outputs no bits must be set. Applications shall not interpret this as lack of audio support. Drivers may automatically select audio outputs without enumerating them. For details on audio outputs and how to select the current output see *Audio Inputs and Outputs* .
__u32	modulator	Output devices can have zero or more RF modulators. When the type is V4L2_OUTPUT_TYPE_MODULATOR this is an RF connector and this field identifies the modulator. It corresponds to struct *v4l2_modulator* field index. For details on modulators see *Tuners and Modulators* .
v4l2_std_id	std	Every video output supports one or more different video standards. This field is a set of all supported standards. For details on video standards and how to switch see *Video Standards* .
__u32	capabilities	This field provides capabilities for the output. See *Output capabilities* for flags.
__u32	reserved[3]	Reserved for future extensions. Drivers must set the array to zero.

Table 1.91: Output Type

V4L2_OUTPUT_TYPE_MODULATOR	1	This output is an analog TV modulator.
V4L2_OUTPUT_TYPE_ANALOG	2	Analog baseband output, for example Composite / CVBS, S-Video, RGB.
V4L2_OUTPUT_TYPE_ANALOGVGAOVERLAY	3	[?]

Table 1.92: Output capabilities

V4L2_OUT_CAP_DV_TIMINGS	0x00000002	This output supports setting video timings by using VIDIOC_S_DV_TIMINGS.
V4L2_OUT_CAP_STD	0x00000004	This output supports setting the TV standard by using VIDIOC_S_STD.
V4L2_OUT_CAP_NATIVE_SIZE	0x00000008	This output supports setting the native size using the V4L2_SEL_TGT_NATIVE_SIZE selection target, see *Common selection definitions* .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_output* `index` is out of bounds.

ioctl VIDIOC_ENUMSTD

Name

VIDIOC_ENUMSTD - Enumerate supported video standards

Synopsis

int **ioctl**(int *fd*, *VIDIOC_ENUMSTD*, struct *v4l2_standard* *argp)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

To query the attributes of a video standard, especially a custom (driver defined) one, applications initialize the `index` field of struct *v4l2_standard* and call the *ioctl VIDIOC_ENUMSTD* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all standards applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`. Drivers may enumerate a different set of standards after switching the video input or output. [1]

v4l2_standard

[1] The supported standards may overlap and we need an unambiguous set to find the current standard returned by *VIDIOC_G_STD* .

Table 1.93: struct v4l2_standard

__u32	index	Number of the video standard, set by the application.
v4l2_std_id	id	The bits in this field identify the standard as one of the common standards listed in *typedef v4l2_std_id* , or if bits 32 to 63 are set as custom standards. Multiple bits can be set if the hardware does not distinguish between these standards, however separate indices do not indicate the opposite. The id must be unique. No other enumerated struct *v4l2_standard* structure, for this input or output anyway, can contain the same set of bits.
__u8	name[24]	Name of the standard, a NUL-terminated ASCII string, for example: "PAL-B/G", "NTSC Japan". This information is intended for the user.
struct *v4l2_fract*	frameperiod	The frame period (not field period) is numerator / denominator. For example M/NTSC has a frame period of 1001 / 30000 seconds.
__u32	framelines	Total lines per frame including blanking, e. g. 625 for B/PAL.
__u32	reserved[4]	Reserved for future extensions. Drivers must set the array to zero.

v4l2_fract

Table 1.94: struct v4l2_fract

__u32	numerator	
__u32	denominator	

Table 1.95: typedef v4l2_std_id

__u64	v4l2_std_id	This type is a set, each bit representing another video standard as listed below and in *Video Standards (based on itu470)* . The 32 most significant bits are reserved for custom (driver defined) video standards.

```
#define V4L2_STD_PAL_B          ((v4l2_std_id)0x00000001)
#define V4L2_STD_PAL_B1         ((v4l2_std_id)0x00000002)
#define V4L2_STD_PAL_G          ((v4l2_std_id)0x00000004)
#define V4L2_STD_PAL_H          ((v4l2_std_id)0x00000008)
#define V4L2_STD_PAL_I          ((v4l2_std_id)0x00000010)
#define V4L2_STD_PAL_D          ((v4l2_std_id)0x00000020)
#define V4L2_STD_PAL_D1         ((v4l2_std_id)0x00000040)
#define V4L2_STD_PAL_K          ((v4l2_std_id)0x00000080)

#define V4L2_STD_PAL_M          ((v4l2_std_id)0x00000100)
#define V4L2_STD_PAL_N          ((v4l2_std_id)0x00000200)
#define V4L2_STD_PAL_Nc         ((v4l2_std_id)0x00000400)
#define V4L2_STD_PAL_60         ((v4l2_std_id)0x00000800)
```

V4L2_STD_PAL_60 is a hybrid standard with 525 lines, 60 Hz refresh rate, and PAL color modulation with a 4.43 MHz color subcarrier. Some PAL video recorders can play back NTSC tapes in this mode for display on a 50/60 Hz agnostic PAL TV.

```
#define V4L2_STD_NTSC_M         ((v4l2_std_id)0x00001000)
#define V4L2_STD_NTSC_M_JP      ((v4l2_std_id)0x00002000)
```

```
#define V4L2_STD_NTSC_443       ((v4l2_std_id)0x00004000)
```

V4L2_STD_NTSC_443 is a hybrid standard with 525 lines, 60 Hz refresh rate, and NTSC color modulation with a 4.43 MHz color subcarrier.

```
#define V4L2_STD_NTSC_M_KR      ((v4l2_std_id)0x00008000)

#define V4L2_STD_SECAM_B        ((v4l2_std_id)0x00010000)
#define V4L2_STD_SECAM_D        ((v4l2_std_id)0x00020000)
#define V4L2_STD_SECAM_G        ((v4l2_std_id)0x00040000)
#define V4L2_STD_SECAM_H        ((v4l2_std_id)0x00080000)
#define V4L2_STD_SECAM_K        ((v4l2_std_id)0x00100000)
#define V4L2_STD_SECAM_K1       ((v4l2_std_id)0x00200000)
#define V4L2_STD_SECAM_L        ((v4l2_std_id)0x00400000)
#define V4L2_STD_SECAM_LC       ((v4l2_std_id)0x00800000)

/* ATSC/HDTV */
#define V4L2_STD_ATSC_8_VSB     ((v4l2_std_id)0x01000000)
#define V4L2_STD_ATSC_16_VSB    ((v4l2_std_id)0x02000000)
```

V4L2_STD_ATSC_8_VSB and V4L2_STD_ATSC_16_VSB are U.S. terrestrial digital TV standards. Presently the V4L2 API does not support digital TV. See also the Linux DVB API at https://linuxtv.org.

```
#define V4L2_STD_PAL_BG         (V4L2_STD_PAL_B         |
                V4L2_STD_PAL_B1         |
                V4L2_STD_PAL_G)
#define V4L2_STD_B              (V4L2_STD_PAL_B         |
                V4L2_STD_PAL_B1         |
                V4L2_STD_SECAM_B)
#define V4L2_STD_GH             (V4L2_STD_PAL_G         |
                V4L2_STD_PAL_H          |
                V4L2_STD_SECAM_G        |
                V4L2_STD_SECAM_H)
#define V4L2_STD_PAL_DK         (V4L2_STD_PAL_D         |
                V4L2_STD_PAL_D1         |
                V4L2_STD_PAL_K)
#define V4L2_STD_PAL            (V4L2_STD_PAL_BG        |
                V4L2_STD_PAL_DK         |
                V4L2_STD_PAL_H          |
                V4L2_STD_PAL_I)
#define V4L2_STD_NTSC           (V4L2_STD_NTSC_M        |
                V4L2_STD_NTSC_M_JP      |
                V4L2_STD_NTSC_M_KR)
#define V4L2_STD_MN             (V4L2_STD_PAL_M         |
                V4L2_STD_PAL_N          |
                V4L2_STD_PAL_Nc         |
                V4L2_STD_NTSC)
#define V4L2_STD_SECAM_DK       (V4L2_STD_SECAM_D       |
                V4L2_STD_SECAM_K        |
                V4L2_STD_SECAM_K1)
#define V4L2_STD_DK             (V4L2_STD_PAL_DK        |
                V4L2_STD_SECAM_DK)

#define V4L2_STD_SECAM          (V4L2_STD_SECAM_B       |
                V4L2_STD_SECAM_G        |
                V4L2_STD_SECAM_H        |
                V4L2_STD_SECAM_DK       |
                V4L2_STD_SECAM_L        |
                V4L2_STD_SECAM_LC)

#define V4L2_STD_525_60         (V4L2_STD_PAL_M         |
                V4L2_STD_PAL_60         |
```

```
                V4L2_STD_NTSC           |
                V4L2_STD_NTSC_443)
#define V4L2_STD_625_50         (V4L2_STD_PAL           |
                V4L2_STD_PAL_N          |
                V4L2_STD_PAL_Nc         |
                V4L2_STD_SECAM)

#define V4L2_STD_UNKNOWN        0
#define V4L2_STD_ALL            (V4L2_STD_525_60        |
                V4L2_STD_625_50)
```

Table 1.96: Video Standards (based on ITU BT.470)

Characteristics	M/NTSC [2]	M/PAL	N/PAL [3]	B, B1, G/PAL	D, D1, K/PAL	H/PAL	I/PAL	B, G/SECAM	D, K/SECAM	K1/SECAM	L/SECAM
Frame lines	525		625								
Frame period (s)	1001/30000		1/25								
Chrominance sub-carrier frequency (Hz)	3579545 ± 10	3579611.49 ± 10	4433618.75 ± 5 (3582056.25 ± 5)	4433618.75 ± 5				4433618.75 ± 1	f_{OR} = 4406250 ± 2000, f_{OB} = 4250000 ± 2000		
Nominal radio-frequency channel bandwidth (MHz)	6	6	6	B: 7; B1, G: 8	8	8	8	8	8	8	8
Sound carrier relative to vision carrier (MHz)	4.5	4.5	4.5	5.5 ± 0.001 [4] [5] [6] [7]	6.5 ± 0.001	5.5	5.9996 ± 0.0005	5.5 ± 0.001	6.5 ± 0.001	6.5	6.5 [8]

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_standard* `index` is out of bounds.

ENODATA Standard video timings are not supported for this input or output.

ioctl VIDIOC_EXPBUF

Name

VIDIOC_EXPBUF - Export a buffer as a DMABUF file descriptor.

Synopsis

int **ioctl**(int *fd*, *VIDIOC_EXPBUF*, struct *v4l2_exportbuffer* *argp)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

This ioctl is an extension to the *memory mapping* I/O method, therefore it is available only for `V4L2_MEMORY_MMAP` buffers. It can be used to export a buffer as a DMABUF file at any time after buffers have been allocated with the *ioctl VIDIOC_REQBUFS* ioctl.

To export a buffer, applications fill struct *v4l2_exportbuffer*. The type field is set to the same buffer type as was previously used with struct *v4l2_requestbuffers* type. Applications must also set the index field. Valid index numbers range from zero to the number of buffers allocated with *ioctl VIDIOC_REQBUFS* (struct *v4l2_requestbuffers* count) minus one. For the multi-planar API, applications set the plane field to the index of the plane to be exported. Valid planes range from zero to the maximal number of valid planes for the currently active format. For the single-planar API, applications must set plane to zero. Additional flags may be posted in the flags field. Refer to a manual for open() for details. Currently only O_CLOEXEC, O_RDONLY, O_WRONLY, and O_RDWR are supported. All other fields must be set to zero. In the case of multi-planar API, every plane is exported separately using multiple *ioctl VIDIOC_EXPBUF* calls.

After calling *ioctl VIDIOC_EXPBUF* the fd field will be set by a driver. This is a DMABUF file descriptor. The application may pass it to other DMABUF-aware devices. Refer to *DMABUF importing* for details about importing DMABUF files into V4L2 nodes. It is recommended to close a DMABUF file when it is no longer used to allow the associated memory to be reclaimed.

Examples

```
int buffer_export(int v4lfd, enum v4l2_buf_type bt, int index, int *dmafd)
{
    struct v4l2_exportbuffer expbuf;

    memset(&expbuf, 0, sizeof(expbuf));
    expbuf.type = bt;
    expbuf.index = index;
    if (ioctl(v4lfd, VIDIOC_EXPBUF, &expbuf) == -1) {
        perror("VIDIOC_EXPBUF");
        return -1;
    }

    *dmafd = expbuf.fd;

    return 0;
}
```

```
int buffer_export_mp(int v4lfd, enum v4l2_buf_type bt, int index,
    int dmafd[], int n_planes)
{
    int i;

    for (i = 0; i < n_planes; ++i) {
        struct v4l2_exportbuffer expbuf;

        memset(&expbuf, 0, sizeof(expbuf));
        expbuf.type = bt;
        expbuf.index = index;
        expbuf.plane = i;
        if (ioctl(v4lfd, VIDIOC_EXPBUF, &expbuf) == -1) {
            perror("VIDIOC_EXPBUF");
            while (i)
                close(dmafd[--i]);
            return -1;
        }
        dmafd[i] = expbuf.fd;
    }

    return 0;
}
```

v4l2_exportbuffer

Table 1.97: struct v4l2_exportbuffer

__u32	type	Type of the buffer, same as struct *v4l2_format* type or struct *v4l2_requestbuffers* type, set by the application. See *v4l2_buf_type*
__u32	index	Number of the buffer, set by the application. This field is only used for *memory mapping* I/O and can range from zero to the number of buffers allocated with the *ioctl VIDIOC_REQBUFS* and/or *ioctl VIDIOC_CREATE_BUFS* ioctls.
__u32	plane	Index of the plane to be exported when using the multi-planar API. Otherwise this value must be set to zero.
__u32	flags	Flags for the newly created file, currently only O_CLOEXEC, O_RDONLY, O_WRONLY, and O_RDWR are supported, refer to the manual of open() for more details.
__s32	fd	The DMABUF file descriptor associated with a buffer. Set by the driver.
__u32	reserved[11]	Reserved field for future use. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL A queue is not in MMAP mode or DMABUF exporting is not supported or flags or type or index or plane fields are invalid.

ioctl VIDIOC_G_AUDIO, VIDIOC_S_AUDIO

Name

VIDIOC_G_AUDIO - VIDIOC_S_AUDIO - Query or select the current audio input and its attributes

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_AUDIO*, struct *v4l2_audio* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_AUDIO*, const struct *v4l2_audio* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the current audio input applications zero out the reserved array of a struct *v4l2_audio* and call the *VIDIOC_G_AUDIO* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code when the device has no audio inputs, or none which combine with the current video input.

Audio inputs have one writable property, the audio mode. To select the current audio input *and* change the audio mode, applications initialize the `index` and `mode` fields, and the `reserved` array of a struct *v4l2_audio* structure and call the *VIDIOC_S_AUDIO* ioctl. Drivers may switch to a different audio mode if the request cannot be satisfied. However, this is a write-only ioctl, it does not return the actual new audio mode.

v4l2_audio

Table 1.98: struct v4l2_audio

__u32	index	Identifies the audio input, set by the driver or application.
__u8	name[32]	Name of the audio input, a NUL-terminated ASCII string, for example: "Line In". This information is intended for the user, preferably the connector label on the device itself.
__u32	capability	Audio capability flags, see *Audio Capability Flags* .
__u32	mode	Audio mode flags set by drivers and applications (on *VIDIOC_S_AUDIO* ioctl), see *Audio Mode Flags* .
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 1.99: Audio Capability Flags

V4L2_AUDCAP_STEREO	0x00001	This is a stereo input. The flag is intended to automatically disable stereo recording etc. when the signal is always monaural. The API provides no means to detect if stereo is *received*, unless the audio input belongs to a tuner.
V4L2_AUDCAP_AVL	0x00002	Automatic Volume Level mode is supported.

Table 1.100: Audio Mode Flags

V4L2_AUDMODE_AVL	0x00001	AVL mode is on.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL No audio inputs combine with the current video input, or the number of the selected audio input is out of bounds or it does not combine.

ioctl VIDIOC_G_AUDOUT, VIDIOC_S_AUDOUT

Name

VIDIOC_G_AUDOUT - VIDIOC_S_AUDOUT - Query or select the current audio output

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_AUDOUT*, struct *v4l2_audioout* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_AUDOUT*, const struct *v4l2_audioout* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the current audio output applications zero out the reserved array of a struct *v4l2_audioout* and call the VIDIOC_G_AUDOUT ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code when the device has no audio inputs, or none which combine with the current video output.

Audio outputs have no writable properties. Nevertheless, to select the current audio output applications can initialize the index field and reserved array (which in the future may contain writable properties) of a struct *v4l2_audioout* structure and call the VIDIOC_S_AUDOUT ioctl. Drivers switch to the requested output or return the EINVAL error code when the index is out of bounds. This is a write-only ioctl, it does not return the current audio output attributes as VIDIOC_G_AUDOUT does.

Note:

Connectors on a TV card to loop back the received audio signal to a sound card are not audio outputs in this sense.

v4l2_audioout

Table 1.101: struct v4l2_audioout

__u32	index	Identifies the audio output, set by the driver or application.
__u8	name[32]	Name of the audio output, a NUL-terminated ASCII string, for example: “Line Out”. This information is intended for the user, preferably the connector label on the device itself.
__u32	capability	Audio capability flags, none defined yet. Drivers must set this field to zero.
__u32	mode	Audio mode, none defined yet. Drivers and applications (on VIDIOC_S_AUDOUT) must set this field to zero.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL No audio outputs combine with the current video output, or the number of the selected audio output is out of bounds or it does not combine.

ioctl VIDIOC_G_CROP, VIDIOC_S_CROP

Name

VIDIOC_G_CROP - VIDIOC_S_CROP - Get or set the current cropping rectangle

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_CROP*, struct *v4l2_crop* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_CROP*, const struct *v4l2_crop* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the cropping rectangle size and position applications set the type field of a struct *v4l2_crop* structure to the respective buffer (stream) type and call the *VIDIOC_G_CROP* ioctl with a pointer to this structure. The driver fills the rest of the structure or returns the EINVAL error code if cropping is not supported.

To change the cropping rectangle applications initialize the type and struct *v4l2_rect* substructure named c of a v4l2_crop structure and call the *VIDIOC_S_CROP* ioctl with a pointer to this structure.

Do not use the multiplanar buffer types. Use V4L2_BUF_TYPE_VIDEO_CAPTURE instead of V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE and use V4L2_BUF_TYPE_VIDEO_OUTPUT instead of V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE.

The driver first adjusts the requested dimensions against hardware limits, i. e. the bounds given by the capture/output window, and it rounds to the closest possible values of horizontal and vertical offset, width and height. In particular the driver must round the vertical offset of the cropping rectangle to frame lines modulo two, such that the field order cannot be confused.

Second the driver adjusts the image size (the opposite rectangle of the scaling process, source or target depending on the data direction) to the closest size possible while maintaining the current horizontal and vertical scaling factor.

Finally the driver programs the hardware with the actual cropping and image parameters. *VIDIOC_S_CROP* is a write-only ioctl, it does not return the actual parameters. To query them applications must call *VIDIOC_G_CROP* and *ioctl VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT* . When the parameters are unsuitable the application may modify the cropping or image parameters and repeat the cycle until satisfactory parameters have been negotiated.

When cropping is not supported then no parameters are changed and *VIDIOC_S_CROP* returns the EINVAL error code.

v4l2_crop

Table 1.102: struct v4l2_crop

__u32	type	Type of the data stream, set by the application. Only these types are valid here: V4L2_BUF_TYPE_VIDEO_CAPTURE, V4L2_BUF_TYPE_VIDEO_OUTPUT and V4L2_BUF_TYPE_VIDEO_OVERLAY. See *v4l2_buf_type*.
struct *v4l2_rect*	c	Cropping rectangle. The same co-ordinate system as for struct *v4l2_cropcap* bounds is used.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA Cropping is not supported for this input or output.

ioctl VIDIOC_G_CTRL, VIDIOC_S_CTRL

Name

VIDIOC_G_CTRL - VIDIOC_S_CTRL - Get or set the value of a control

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_CTRL*, struct *v4l2_control* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_CTRL*, struct *v4l2_control* *argp)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

To get the current value of a control applications initialize the `id` field of a struct *`v4l2_control`* and call the *VIDIOC_G_CTRL* ioctl with a pointer to this structure. To change the value of a control applications initialize the `id` and `value` fields of a struct *`v4l2_control`* and call the *VIDIOC_S_CTRL* ioctl.

When the `id` is invalid drivers return an `EINVAL` error code. When the `value` is out of bounds drivers can choose to take the closest valid value or return an `ERANGE` error code, whatever seems more appropriate. However, *VIDIOC_S_CTRL* is a write-only ioctl, it does not return the actual new value. If the `value` is inappropriate for the control (e.g. if it refers to an unsupported menu index of a menu control), then EINVAL error code is returned as well.

These ioctls work only with user controls. For other control classes the *VIDIOC_G_EXT_CTRLS* , *VIDIOC_S_EXT_CTRLS* or *VIDIOC_TRY_EXT_CTRLS* must be used.

`v4l2_control`

Table 1.103: struct v4l2_control

__u32	`id`	Identifies the control, set by the application.
__s32	`value`	New value or current value.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *`v4l2_control`* `id` is invalid or the `value` is inappropriate for the given control (i.e. if a menu item is selected that is not supported by the driver according to *VIDIOC_QUERYMENU*).

ERANGE The struct *`v4l2_control`* `value` is out of bounds.

EBUSY The control is temporarily not changeable, possibly because another applications took over control of the device function this control belongs to.

EACCES Attempt to set a read-only control or to get a write-only control.

ioctl VIDIOC_G_DV_TIMINGS, VIDIOC_S_DV_TIMINGS

Name

VIDIOC_G_DV_TIMINGS - VIDIOC_S_DV_TIMINGS - VIDIOC_SUBDEV_G_DV_TIMINGS - VIDIOC_SUBDEV_S_DV_TIMINGS - Get or set DV timings for input or output

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_DV_TIMINGS*, struct *v4l2_dv_timings* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_DV_TIMINGS*, struct *v4l2_dv_timings* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_G_DV_TIMINGS*, struct *v4l2_dv_timings* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_S_DV_TIMINGS*, struct *v4l2_dv_timings* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To set DV timings for the input or output, applications use the *VIDIOC_S_DV_TIMINGS* ioctl and to get the current timings, applications use the *VIDIOC_G_DV_TIMINGS* ioctl. The detailed timing information is filled in using the structure struct *v4l2_dv_timings*. These ioctls take a pointer to the struct *v4l2_dv_timings* structure as argument. If the ioctl is not supported or the timing values are not correct, the driver returns EINVAL error code.

The linux/v4l2-dv-timings.h header can be used to get the timings of the formats in the *CEA-861-E* and *VESA DMT* standards. If the current input or output does not support DV timings (e.g. if *ioctl VIDIOC_ENUMINPUT* does not set the V4L2_IN_CAP_DV_TIMINGS flag), then ENODATA error code is returned.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL This ioctl is not supported, or the *VIDIOC_S_DV_TIMINGS* parameter was unsuitable.

ENODATA Digital video timings are not supported for this input or output.

EBUSY The device is busy and therefore can not change the timings.

v4l2_bt_timings

Table 1.104: struct v4l2_bt_timings

__u32	width	Width of the active video in pixels.
__u32	height	Height of the active video frame in lines. So for interlaced formats the height of the active video in each field is height/2.
__u32	interlaced	Progressive (V4L2_DV_PROGRESSIVE) or interlaced (V4L2_DV_INTERLACED).
__u32	polarities	This is a bit mask that defines polarities of sync signals. bit 0 (V4L2_DV_VSYNC_POS_POL) is for vertical sync polarity and bit 1 (V4L2_DV_HSYNC_POS_POL) is for horizontal sync polarity. If the bit is set (1) it is positive polarity and if is cleared (0), it is negative polarity.
__u64	pixelclock	Pixel clock in Hz. Ex. 74.25MHz->74250000
__u32	hfrontporch	Horizontal front porch in pixels
__u32	hsync	Horizontal sync length in pixels
__u32	hbackporch	Horizontal back porch in pixels
__u32	vfrontporch	Vertical front porch in lines. For interlaced formats this refers to the odd field (aka field 1).
__u32	vsync	Vertical sync length in lines. For interlaced formats this refers to the odd field (aka field 1).
__u32	vbackporch	Vertical back porch in lines. For interlaced formats this refers to the odd field (aka field 1).
__u32	il_vfrontporch	Vertical front porch in lines for the even field (aka field 2) of interlaced field formats. Must be 0 for progressive formats.
__u32	il_vsync	Vertical sync length in lines for the even field (aka field 2) of interlaced field formats. Must be 0 for progressive formats.
__u32	il_vbackporch	Vertical back porch in lines for the even field (aka field 2) of interlaced field formats. Must be 0 for progressive formats.
__u32	standards	The video standard(s) this format belongs to. This will be filled in by the driver. Applications must set this to 0. See *DV BT Timing standards* for a list of standards.
__u32	flags	Several flags giving more information about the format. See *DV BT Timing flags* for a description of the flags.
__u32	reserved[14]	Reserved for future extensions. Drivers and applications must set the array to zero.

v4l2_dv_timings

Table 1.105: struct v4l2_dv_timings

__u32	type		Type of DV timings as listed in *DV Timing types* .
union			
	struct *v4l2_bt_timings*	bt	Timings defined by BT.656/1120 specifications
	__u32	reserved[32]	

Table 1.106: DV Timing types

Timing type	value	Description
V4L2_DV_BT_656_1120	0	BT.656/1120 timings

Table 1.107: DV BT Timing standards

Timing standard	Description
V4L2_DV_BT_STD_CEA861	The timings follow the CEA-861 Digital TV Profile standard
V4L2_DV_BT_STD_DMT	The timings follow the VESA Discrete Monitor Timings standard
V4L2_DV_BT_STD_CVT	The timings follow the VESA Coordinated Video Timings standard
V4L2_DV_BT_STD_GTF	The timings follow the VESA Generalized Timings Formula standard
V4L2_DV_BT_STD_SDI	The timings follow the SDI Timings standard. There are no horizontal syncs/porches at all in this format. Total blanking timings must be set in hsync or vsync fields only.

Table 1.108: DV BT Timing flags

Flag	Description
V4L2_DV_FL_REDUCED_BLANKING	CVT/GTF specific: the timings use reduced blanking (CVT) or the 'Secondary GTF' curve (GTF). In both cases the horizontal and/or vertical blanking intervals are reduced, allowing a higher resolution over the same bandwidth. This is a read-only flag, applications must not set this.
V4L2_DV_FL_CAN_REDUCE_FPS	CEA-861 specific: set for CEA-861 formats with a framerate that is a multiple of six. These formats can be optionally played at 1 / 1.001 speed to be compatible with 60 Hz based standards such as NTSC and PAL-M that use a framerate of 29.97 frames per second. If the transmitter can't generate such frequencies, then the flag will also be cleared. This is a read-only flag, applications must not set this.
V4L2_DV_FL_REDUCED_FPS	CEA-861 specific: only valid for video transmitters, the flag is cleared by receivers. It is also only valid for formats with the V4L2_DV_FL_CAN_REDUCE_FPS flag set, for other formats the flag will be cleared by the driver. If the application sets this flag, then the pixelclock used to set up the transmitter is divided by 1.001 to make it compatible with NTSC framerates. If the transmitter can't generate such frequencies, then the flag will also be cleared.
V4L2_DV_FL_HALF_LINE	Specific to interlaced formats: if set, then the vertical frontporch of field 1 (aka the odd field) is really one half-line longer and the vertical backporch of field 2 (aka the even field) is really one half-line shorter, so each field has exactly the same number of half-lines. Whether half-lines can be detected or used depends on the hardware.
V4L2_DV_FL_IS_CE_VIDEO	If set, then this is a Consumer Electronics (CE) video format. Such formats differ from other formats (commonly called IT formats) in that if R'G'B' encoding is used then by default the R'G'B' values use limited range (i.e. 16-235) as opposed to full range (i.e. 0-255). All formats defined in CEA-861 except for the 640x480p59.94 format are CE formats.
V4L2_DV_FL_FIRST_FIELD_EXTRA_LINE	Some formats like SMPTE-125M have an interlaced signal with a odd total height. For these formats, if this flag is set, the first field has the extra line. Else, it is the second field.
V4L2_DV_FL_HAS_PICTURE_ASPECT	If set, then the picture_aspect field is valid. Otherwise assume that the pixels are square, so the picture aspect ratio is the same as the width to height ratio.
V4L2_DV_FL_HAS_CEA861_VIC	If set, then the cea861_vic field is valid and contains the Video Identification Code as per the CEA-861 standard.
V4L2_DV_FL_HAS_HDMI_VIC	If set, then the hdmi_vic field is valid and contains the Video Identification Code as per the HDMI standard (HDMI Vendor Specific InfoFrame).

ioctl VIDIOC_G_EDID, VIDIOC_S_EDID, VIDIOC_SUBDEV_G_EDID, VIDIOC_SUBDEV_S_EDID

Name

VIDIOC_G_EDID - VIDIOC_S_EDID - VIDIOC_SUBDEV_G_EDID - VIDIOC_SUBDEV_S_EDID - Get or set the EDID of a video receiver/transmitter

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_EDID*, struct *v4l2_edid* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_EDID*, struct *v4l2_edid* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_G_EDID*, struct *v4l2_edid* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_S_EDID*, struct *v4l2_edid* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

These ioctls can be used to get or set an EDID associated with an input from a receiver or an output of a transmitter device. They can be used with subdevice nodes (/dev/v4l-subdevX) or with video nodes (/dev/videoX).

When used with video nodes the pad field represents the input (for video capture devices) or output (for video output devices) index as is returned by *ioctl VIDIOC_ENUMINPUT* and *ioctl VIDIOC_ENUMOUTPUT* respectively. When used with subdevice nodes the pad field represents the input or output pad of the subdevice. If there is no EDID support for the given pad value, then the EINVAL error code will be returned.

To get the EDID data the application has to fill in the pad, start_block, blocks and edid fields, zero the reserved array and call *VIDIOC_G_EDID* . The current EDID from block start_block and of size blocks will be placed in the memory edid points to. The edid pointer must point to memory at least blocks * 128 bytes large (the size of one block is 128 bytes).

If there are fewer blocks than specified, then the driver will set blocks to the actual number of blocks. If there are no EDID blocks available at all, then the error code ENODATA is set.

If blocks have to be retrieved from the sink, then this call will block until they have been read.

If start_block and blocks are both set to 0 when *VIDIOC_G_EDID* is called, then the driver will set blocks to the total number of available EDID blocks and it will return 0 without copying any data. This is an easy way to discover how many EDID blocks there are.

Note:

If there are no EDID blocks available at all, then the driver will set blocks to 0 and it returns 0.

To set the EDID blocks of a receiver the application has to fill in the pad, blocks and edid fields, set start_block to 0 and zero the reserved array. It is not possible to set part of an EDID, it is always all or nothing. Setting the EDID data is only valid for receivers as it makes no sense for a transmitter.

The driver assumes that the full EDID is passed in. If there are more EDID blocks than the hardware can handle then the EDID is not written, but instead the error code E2BIG is set and blocks is set to the maximum that the hardware supports. If start_block is any value other than 0 then the error code EINVAL is set.

To disable an EDID you set blocks to 0. Depending on the hardware this will drive the hotplug pin low and/or block the source from reading the EDID data in some way. In any case, the end result is the same: the EDID is no longer available.

v4l2_edid

Table 1.109: struct v4l2_edid

__u32	pad	Pad for which to get/set the EDID blocks. When used with a video device node the pad represents the input or output index as returned by *ioctl VIDIOC_ENUMINPUT* and *ioctl VIDIOC_ENUMOUTPUT* respectively.
__u32	start_block	Read the EDID from starting with this block. Must be 0 when setting the EDID.
__u32	blocks	The number of blocks to get or set. Must be less or equal to 256 (the maximum number of blocks as defined by the standard). When you set the EDID and blocks is 0, then the EDID is disabled or erased.
__u32	reserved[5]	Reserved for future extensions. Applications and drivers must set the array to zero.
__u8 *	edid	Pointer to memory that contains the EDID. The minimum size is blocks * 128.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA The EDID data is not available.

E2BIG The EDID data you provided is more than the hardware can handle.

ioctl VIDIOC_G_ENC_INDEX

Name

VIDIOC_G_ENC_INDEX - Get meta data about a compressed video stream

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_ENC_INDEX*, struct *v4l2_enc_idx* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

The *VIDIOC_G_ENC_INDEX* ioctl provides meta data about a compressed video stream the same or another application currently reads from the driver, which is useful for random access into the stream without decoding it.

To read the data applications must call *VIDIOC_G_ENC_INDEX* with a pointer to a struct *v4l2_enc_idx*. On success the driver fills the entry array, stores the number of elements written in the entries field, and initializes the entries_cap field.

Each element of the entry array contains meta data about one picture. A *VIDIOC_G_ENC_INDEX* call reads up to V4L2_ENC_IDX_ENTRIES entries from a driver buffer, which can hold up to entries_cap entries. This number can be lower or higher than V4L2_ENC_IDX_ENTRIES, but not zero. When the application fails to read the meta data in time the oldest entries will be lost. When the buffer is empty or no capturing/encoding is in progress, entries will be zero.

Currently this ioctl is only defined for MPEG-2 program streams and video elementary streams.

v4l2_enc_idx

Table 1.110: struct v4l2_enc_idx

__u32	entries	The number of entries the driver stored in the entry array.
__u32	entries_cap	The number of entries the driver can buffer. Must be greater than zero.
__u32	reserved[4]	Reserved for future extensions. Drivers must set the array to zero.
struct *v4l2_enc_idx_entry*	entry[V4L2_ENC_IDX_ENTRIES]	Meta data about a compressed video stream. Each element of the array corresponds to one picture, sorted in ascending order by their offset.

v4l2_enc_idx_entry

Table 1.111: struct v4l2_enc_idx_entry

__u64	offset	The offset in bytes from the beginning of the compressed video stream to the beginning of this picture, that is a *PES packet header* as defined in *ISO 13818-1* or a *picture header* as defined in *ISO 13818-2* . When the encoder is stopped, the driver resets the offset to zero.
__u64	pts	The 33 bit *Presentation Time Stamp* of this picture as defined in *ISO 13818-1* .
__u32	length	The length of this picture in bytes.
__u32	flags	Flags containing the coding type of this picture, see *Index Entry Flags* .
__u32	reserved[2]	Reserved for future extensions. Drivers must set the array to zero.

Table 1.112: Index Entry Flags

V4L2_ENC_IDX_FRAME_I	0x00	This is an Intra-coded picture.
V4L2_ENC_IDX_FRAME_P	0x01	This is a Predictive-coded picture.
V4L2_ENC_IDX_FRAME_B	0x02	This is a Bidirectionally predictive-coded picture.
V4L2_ENC_IDX_FRAME_MASK	0x0F	*AND* the flags field with this mask to obtain the picture coding type.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_G_EXT_CTRLS, VIDIOC_S_EXT_CTRLS, VIDIOC_TRY_EXT_CTRLS

Name

VIDIOC_G_EXT_CTRLS - VIDIOC_S_EXT_CTRLS - VIDIOC_TRY_EXT_CTRLS - Get or set the value of several controls, try control values

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_EXT_CTRLS*, struct *v4l2_ext_controls* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_EXT_CTRLS*, struct *v4l2_ext_controls* *argp)

int **ioctl**(int *fd*, *VIDIOC_TRY_EXT_CTRLS*, struct *v4l2_ext_controls* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

These ioctls allow the caller to get or set multiple controls atomically. Control IDs are grouped into control classes (see *Control classes*) and all controls in the control array must belong to the same control class.

Applications must always fill in the count, which, controls and reserved fields of struct *v4l2_ext_controls*, and initialize the struct *v4l2_ext_control* array pointed to by the controls fields.

To get the current value of a set of controls applications initialize the id, size and reserved2 fields of each struct *v4l2_ext_control* and call the *VIDIOC_G_EXT_CTRLS* ioctl. String controls controls must also set the string field. Controls of compound types (V4L2_CTRL_FLAG_HAS_PAYLOAD is set) must set the ptr field.

If the size is too small to receive the control result (only relevant for pointer-type controls like strings), then the driver will set size to a valid value and return an ENOSPC error code. You should re-allocate the memory to this new size and try again. For the string type it is possible that the same issue occurs again if the string has grown in the meantime. It is recommended to call *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* first and use maximum+1 as the new size value. It is guaranteed that that is sufficient memory.

N-dimensional arrays are set and retrieved row-by-row. You cannot set a partial array, all elements have to be set or retrieved. The total size is calculated as elems * elem_size. These values can be obtained by calling *VIDIOC_QUERY_EXT_CTRL* .

To change the value of a set of controls applications initialize the id, size, reserved2 and value/value64/string/ptr fields of each struct *v4l2_ext_control* and call the *VIDIOC_S_EXT_CTRLS* ioctl. The controls will only be set if *all* control values are valid.

To check if a set of controls have correct values applications initialize the id, size, reserved2 and value/value64/string/ptr fields of each struct *v4l2_ext_control* and call the *VIDIOC_TRY_EXT_CTRLS* ioctl. It is up to the driver whether wrong values are automatically adjusted to a valid value or if an error is returned.

When the id or which is invalid drivers return an EINVAL error code. When the value is out of bounds drivers can choose to take the closest valid value or return an ERANGE error code, whatever seems more appropriate. In the first case the new value is set in struct *v4l2_ext_control*. If the new control value is inappropriate (e.g. the given menu index is not supported by the menu control), then this will also result in an EINVAL error code error.

The driver will only set/get these controls if all control values are correct. This prevents the situation where only some of the controls were set/get. Only low-level errors (e. g. a failed i2c command) can still cause this situation.

v4l2_ext_control

Table 1.113: struct v4l2_ext_control

__u32	id		Identifies the control, set by the application.
__u32	size		The total size in bytes of the payload of this control. This is normally 0, but for pointer controls this should be set to the size of the memory containing the payload, or that will receive the payload. If *VIDIOC_G_EXT_CTRLS* finds that this value is less than is required to store the payload result, then it is set to a value large enough to store the payload result and ENOSPC is returned. ***Note:*** *For string controls, this size field should not be confused with the length of the string. This field refers to the size of the memory that contains the string. The actual* length *of the string may well be much smaller.*
__u32	reserved2[1]		Reserved for future extensions. Drivers and applications must set the array to zero.
union	(anonymous)		
	__s32	value	New value or current value. Valid if this control is not of type V4L2_CTRL_TYPE_INTEGER64 and V4L2_CTRL_FLAG_HAS_PAYLOAD is not set.
	__s64	value64	New value or current value. Valid if this control is of type V4L2_CTRL_TYPE_INTEGER64 and V4L2_CTRL_FLAG_HAS_PAYLOAD is not set.
	char *	string	A pointer to a string. Valid if this control is of type V4L2_CTRL_TYPE_STRING.
	__u8 *	p_u8	A pointer to a matrix control of unsigned 8-bit values. Valid if this control is of type V4L2_CTRL_TYPE_U8.
	__u16 *	p_u16	A pointer to a matrix control of unsigned 16-bit values. Valid if this control is of type V4L2_CTRL_TYPE_U16.
	__u32 *	p_u32	A pointer to a matrix control of unsigned 32-bit values. Valid if this control is of type V4L2_CTRL_TYPE_U32.
	void *	ptr	A pointer to a compound type which can be an N-dimensional array and/or a compound type (the control's type is >= V4L2_CTRL_COMPOUND_TYPES). Valid if V4L2_CTRL_FLAG_HAS_PAYLOAD is set for this control.

v4l2_ext_controls

Table 1.114: struct v4l2_ext_controls

union	(anonymous)	

Continued on next page

Table 1.114 – continued from previous page

	__u32	ctrl_class	The control class to which all controls belong, see *Control classes* . Drivers that use a kernel framework for handling controls will also accept a value of 0 here, meaning that the controls can belong to any control class. Whether drivers support this can be tested by setting ctrl_class to 0 and calling *VIDIOC_TRY_EXT_CTRLS* with a count of 0. If that succeeds, then the driver supports this feature.
	__u32	which	Which value of the control to get/set/try. V4L2_CTRL_WHICH_CUR_VAL will return the current value of the control and V4L2_CTRL_WHICH_DEF_VAL will return the default value of the control. ***Note:*** *You can only get the default value of the control, you cannot set or try it.* For backwards compatibility you can also use a control class here (see *Control classes*). In that case all controls have to belong to that control class. This usage is deprecated, instead just use V4L2_CTRL_WHICH_CUR_VAL. There are some very old drivers that do not yet support V4L2_CTRL_WHICH_CUR_VAL and that require a control class here. You can test for such drivers by setting ctrl_class to V4L2_CTRL_WHICH_CUR_VAL and calling VIDIOC_TRY_EXT_CTRLS with a count of 0. If that fails, then the driver does not support V4L2_CTRL_WHICH_CUR_VAL.
__u32	count	The number of controls in the controls array. May also be zero.	

Continued on next page

Table 1.114 – continued from previous page

__u32	error_idx	Set by the driver in case of an error. If the error is associated with a particular control, then error_idx is set to the index of that control. If the error is not related to a specific control, or the validation step failed (see below), then error_idx is set to count. The value is undefined if the ioctl returned 0 (success). Before controls are read from/written to hardware a validation step takes place: this checks if all controls in the list are valid controls, if no attempt is made to write to a read-only control or read from a write-only control, and any other up-front checks that can be done without accessing the hardware. The exact validations done during this step are driver dependent since some checks might require hardware access for some devices, thus making it impossible to do those checks up-front. However, drivers should make a best-effort to do as many up-front checks as possible. This check is done to avoid leaving the hardware in an inconsistent state due to easy-to-avoid problems. But it leads to another problem: the application needs to know whether an error came from the validation step (meaning that the hardware was not touched) or from an error during the actual reading from/writing to hardware. The, in hindsight quite poor, solution for that is to set error_idx to count if the validation failed. This has the unfortunate side-effect that it is not possible to see which control failed the validation. If the validation was successful and the error happened while accessing the hardware, then error_idx is less than count and only the controls up to error_idx-1 were read or written correctly, and the state of the remaining controls is undefined. Since *VIDIOC_TRY_EXT_CTRLS* does not access hardware there is also no need to handle the validation step in this special way, so error_idx will just be set to the control that failed the validation step instead of to count. This means that if *VIDIOC_S_EXT_CTRLS* fails with error_idx set to count, then you can call *VIDIOC_TRY_EXT_CTRLS* to try to discover the actual control that failed the validation step. Unfortunately, there is no TRY equivalent for *VIDIOC_G_EXT_CTRLS* .
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.
struct *v4l2_ext_control* *	controls	Pointer to an array of count v4l2_ext_control structures. Ignored if count equals zero.

Table 1.115: Control classes

V4L2_CTRL_CLASS_USER	0x980000	The class containing user controls. These controls are described in *User Controls* . All controls that can be set using the *VIDIOC_S_CTRL* and *VIDIOC_G_CTRL* ioctl belong to this class.
V4L2_CTRL_CLASS_MPEG	0x990000	The class containing MPEG compression controls. These controls are described in *Codec Control Reference* .
V4L2_CTRL_CLASS_CAMERA	0x9a0000	The class containing camera controls. These controls are described in *Camera Control Reference* .
V4L2_CTRL_CLASS_FM_TX	0x9b0000	The class containing FM Transmitter (FM TX) controls. These controls are described in *FM Transmitter Control Reference* .
V4L2_CTRL_CLASS_FLASH	0x9c0000	The class containing flash device controls. These controls are described in *Flash Control Reference* .
V4L2_CTRL_CLASS_JPEG	0x9d0000	The class containing JPEG compression controls. These controls are described in *JPEG Control Reference* .
V4L2_CTRL_CLASS_IMAGE_SOURCE	0x9e0000	The class containing image source controls. These controls are described in *Image Source Control Reference* .
V4L2_CTRL_CLASS_IMAGE_PROC	0x9f0000	The class containing image processing controls. These controls are described in *Image Process Control Reference* .
V4L2_CTRL_CLASS_FM_RX	0xa10000	The class containing FM Receiver (FM RX) controls. These controls are described in *FM Receiver Control Reference* .
V4L2_CTRL_CLASS_RF_TUNER	0xa20000	The class containing RF tuner controls. These controls are described in *RF Tuner Control Reference* .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_ext_control* `id` is invalid, the struct *v4l2_ext_controls* `which` is invalid, or the struct *v4l2_ext_control* `value` was inappropriate (e.g. the given menu index is not supported by the driver). This error code is also returned by the *VIDIOC_S_EXT_CTRLS* and *VIDIOC_TRY_EXT_CTRLS* ioctls if two or more control values are in conflict.

ERANGE The struct *v4l2_ext_control* `value` is out of bounds.

EBUSY The control is temporarily not changeable, possibly because another applications took over control of the device function this control belongs to.

ENOSPC The space reserved for the control's payload is insufficient. The field `size` is set to a value that is enough to store the payload and this error code is returned.

EACCES Attempt to try or set a read-only control or to get a write-only control.

ioctl VIDIOC_G_FBUF, VIDIOC_S_FBUF

Name

VIDIOC_G_FBUF - VIDIOC_S_FBUF - Get or set frame buffer overlay parameters

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_FBUF*, struct *v4l2_framebuffer* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_FBUF*, const struct *v4l2_framebuffer* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Applications can use the *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF* ioctl to get and set the framebuffer parameters for a *Video Overlay* or *Video Output Overlay* (OSD). The type of overlay is implied by the device type (capture or output device) and can be determined with the *ioctl VIDIOC_QUERYCAP* ioctl. One /dev/videoN device must not support both kinds of overlay.

The V4L2 API distinguishes destructive and non-destructive overlays. A destructive overlay copies captured video images into the video memory of a graphics card. A non-destructive overlay blends video images into a VGA signal or graphics into a video signal. *Video Output Overlays* are always non-destructive.

To get the current parameters applications call the *VIDIOC_G_FBUF* ioctl with a pointer to a struct *v4l2_framebuffer* structure. The driver fills all fields of the structure or returns an EINVAL error code when overlays are not supported.

To set the parameters for a *Video Output Overlay*, applications must initialize the flags field of a struct struct *v4l2_framebuffer*. Since the framebuffer is implemented on the TV card all other parameters are determined by the driver. When an application calls *VIDIOC_S_FBUF* with a pointer to this structure, the driver prepares for the overlay and returns the framebuffer parameters as *VIDIOC_G_FBUF* does, or it returns an error code.

To set the parameters for a *non-destructive Video Overlay*, applications must initialize the flags field, the fmt substructure, and call *VIDIOC_S_FBUF* . Again the driver prepares for the overlay and returns the framebuffer parameters as *VIDIOC_G_FBUF* does, or it returns an error code.

For a *destructive Video Overlay* applications must additionally provide a base address. Setting up a DMA to a random memory location can jeopardize the system security, its stability or even damage the hardware, therefore only the superuser can set the parameters for a destructive video overlay.

v4l2_framebuffer

Table 1.116: struct v4l2_framebuffer

__u32	capability		Overlay capability flags set by the driver, see *Frame Buffer Capability Flags* .
__u32	flags		Overlay control flags set by application and driver, see *Frame Buffer Flags*

Continued on next page

Table 1.116 – continued from previous page

void *	base		Physical base address of the framebuffer, that is the address of the pixel in the top left corner of the framebuffer. [1]
			This field is irrelevant to *non-destructive Video Overlays*. For *destructive Video Overlays* applications must provide a base address. The driver may accept only base addresses which are a multiple of two, four or eight bytes. For *Video Output Overlays* the driver must return a valid base address, so applications can find the corresponding Linux framebuffer device (see *Video Output Overlay Interface*).
struct	fmt		Layout of the frame buffer.
	__u32	width	Width of the frame buffer in pixels.
	__u32	height	Height of the frame buffer in pixels.
	__u32	pixelformat	The pixel format of the framebuffer.
			For *non-destructive Video Overlays* this field only defines a format for the struct *v4l2_window* chromakey field.
			For *destructive Video Overlays* applications must initialize this field. For *Video Output Overlays* the driver must return a valid format.
			Usually this is an RGB format (for example *V4L2_PIX_FMT_RGB565*) but YUV formats (only packed YUV formats when chroma keying is used, not including V4L2_PIX_FMT_YUYV and V4L2_PIX_FMT_UYVY) and the V4L2_PIX_FMT_PAL8 format are also permitted. The behavior of the driver when an application requests a compressed format is undefined. See *Image Formats* for information on pixel formats.
	enum *v4l2_field*	field	Drivers and applications shall ignore this field. If applicable, the field order is selected with the *VIDIOC_S_FMT* ioctl, using the field field of struct *v4l2_window*.
	__u32	bytesperline	Distance in bytes between the leftmost pixels in two adjacent lines.

Continued on next page

Table 1.116 – continued from previous page

			This field is irrelevant to *non-destructive Video Overlays*. For *destructive Video Overlays* both applications and drivers can set this field to request padding bytes at the end of each line. Drivers however may ignore the requested value, returning `width` times bytes-per-pixel or a larger value required by the hardware. That implies applications can just set this field to zero to get a reasonable default. For *Video Output Overlays* the driver must return a valid value. Video hardware may access padding bytes, therefore they must reside in accessible memory. Consider for example the case where padding bytes after the last line of an image cross a system page boundary. Capture devices may write padding bytes, the value is undefined. Output devices ignore the contents of padding bytes. When the image format is planar the `bytesperline` value applies to the first plane and is divided by the same factor as the `width` field for the other planes. For example the Cb and Cr planes of a YUV 4:2:0 image have half as many padding bytes following each line as the Y plane. To avoid ambiguities drivers must return a `bytesperline` value rounded up to a multiple of the scale factor.
	__u32	`sizeimage`	This field is irrelevant to *non-destructive Video Overlays*. For *destructive Video Overlays* applications must initialize this field. For *Video Output Overlays* the driver must return a valid format. Together with base it defines the framebuffer memory accessible by the driver.
	enum *v4l2_colorspace*	`colorspace`	This information supplements the `pixelformat` and must be set by the driver, see *Colorspaces* .
	__u32	`priv`	Reserved. Drivers and applications must set this field to zero.

[1] A physical base address may not suit all platforms. GK notes in theory we should pass something like PCI device + memory region + offset instead. If you encounter problems please discuss on the linux-media mailing list: https://linuxtv.org/lists.php.

Table 1.117: Frame Buffer Capability Flags

V4L2_FBUF_CAP_EXTERNOVERLAY	0x0001	The device is capable of non-destructive overlays. When the driver clears this flag, only destructive overlays are supported. There are no drivers yet which support both destructive and non-destructive overlays. Video Output Overlays are in practice always non-destructive.
V4L2_FBUF_CAP_CHROMAKEY	0x0002	The device supports clipping by chroma-keying the images. That is, image pixels replace pixels in the VGA or video signal only where the latter assume a certain color. Chroma-keying makes no sense for destructive overlays.
V4L2_FBUF_CAP_LIST_CLIPPING	0x0004	The device supports clipping using a list of clip rectangles.
V4L2_FBUF_CAP_BITMAP_CLIPPING	0x0008	The device supports clipping using a bit mask.
V4L2_FBUF_CAP_LOCAL_ALPHA	0x0010	The device supports clipping/blending using the alpha channel of the framebuffer or VGA signal. Alpha blending makes no sense for destructive overlays.
V4L2_FBUF_CAP_GLOBAL_ALPHA	0x0020	The device supports alpha blending using a global alpha value. Alpha blending makes no sense for destructive overlays.
V4L2_FBUF_CAP_LOCAL_INV_ALPHA	0x0040	The device supports clipping/blending using the inverted alpha channel of the framebuffer or VGA signal. Alpha blending makes no sense for destructive overlays.
V4L2_FBUF_CAP_SRC_CHROMAKEY	0x0080	The device supports Source Chroma-keying. Video pixels with the chroma-key colors are replaced by framebuffer pixels, which is exactly opposite of V4L2_FBUF_CAP_CHROMAKEY

Table 1.118: Frame Buffer Flags

V4L2_FBUF_FLAG_PRIMARY	0x0001	The framebuffer is the primary graphics surface. In other words, the overlay is destructive. This flag is typically set by any driver that doesn't have the V4L2_FBUF_CAP_EXTERNOVERLAY capability and it is cleared otherwise.
V4L2_FBUF_FLAG_OVERLAY	0x0002	If this flag is set for a video capture device, then the driver will set the initial overlay size to cover the full framebuffer size, otherwise the existing overlay size (as set by *VIDIOC_S_FMT*) will be used. Only one video capture driver (bttv) supports this flag. The use of this flag for capture devices is deprecated. There is no way to detect which drivers support this flag, so the only reliable method of setting the overlay size is through *VIDIOC_S_FMT* . If this flag is set for a video output device, then the video output overlay window is relative to the top-left corner of the framebuffer and restricted to the size of the framebuffer. If it is cleared, then the video output overlay window is relative to the video output display.

Continued on next page

Table 1.118 – continued from previous page

V4L2_FBUF_FLAG_CHROMAKEY	0x0004	Use chroma-keying. The chroma-key color is determined by the chromakey field of struct *v4l2_window* and negotiated with the *VIDIOC_S_FMT* ioctl, see *Video Overlay Interface* and *Video Output Overlay Interface* .
There are no flags to enable clipping using a list of clip rectangles or a bitmap. These methods are negotiated with the *VIDIOC_S_FMT* ioctl, see *Video Overlay Interface* and *Video Output Overlay Interface* .		
V4L2_FBUF_FLAG_LOCAL_ALPHA	0x0008	Use the alpha channel of the framebuffer to clip or blend framebuffer pixels with video images. The blend function is: output = framebuffer pixel * alpha + video pixel * (1 - alpha). The actual alpha depth depends on the framebuffer pixel format.
V4L2_FBUF_FLAG_GLOBAL_ALPHA	0x0010	Use a global alpha value to blend the framebuffer with video images. The blend function is: output = (framebuffer pixel * alpha + video pixel * (255 - alpha)) / 255. The alpha value is determined by the global_alpha field of struct *v4l2_window* and negotiated with the *VIDIOC_S_FMT* ioctl, see *Video Overlay Interface* and *Video Output Overlay Interface* .
V4L2_FBUF_FLAG_LOCAL_INV_ALPHA	0x0020	Like V4L2_FBUF_FLAG_LOCAL_ALPHA, use the alpha channel of the framebuffer to clip or blend framebuffer pixels with video images, but with an inverted alpha value. The blend function is: output = framebuffer pixel * (1 - alpha) + video pixel * alpha. The actual alpha depth depends on the framebuffer pixel format.
V4L2_FBUF_FLAG_SRC_CHROMAKEY	0x0040	Use source chroma-keying. The source chroma-key color is determined by the chromakey field of struct *v4l2_window* and negotiated with the *VIDIOC_S_FMT* ioctl, see *Video Overlay Interface* and *Video Output Overlay Interface* . Both chroma-keying are mutual exclusive to each other, so same chromakey field of struct *v4l2_window* is being used.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EPERM *VIDIOC_S_FBUF* can only be called by a privileged user to negotiate the parameters for a destructive overlay.

EINVAL The *VIDIOC_S_FBUF* parameters are unsuitable.

ioctl VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT

Name

VIDIOC_G_FMT - VIDIOC_S_FMT - VIDIOC_TRY_FMT - Get or set the data format, try a format

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_FMT*, struct *v4l2_format* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_FMT*, struct *v4l2_format* *argp)

int **ioctl**(int *fd*, *VIDIOC_TRY_FMT*, struct *v4l2_format* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

These ioctls are used to negotiate the format of data (typically image format) exchanged between driver and application.

To query the current parameters applications set the type field of a struct *v4l2_format* to the respective buffer (stream) type. For example video capture devices use V4L2_BUF_TYPE_VIDEO_CAPTURE or V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE. When the application calls the *VIDIOC_G_FMT* ioctl with a pointer to this structure the driver fills the respective member of the fmt union. In case of video capture devices that is either the struct *v4l2_pix_format* pix or the struct *v4l2_pix_format_mplane* pix_mp member. When the requested buffer type is not supported drivers return an EINVAL error code.

To change the current format parameters applications initialize the type field and all fields of the respective fmt union member. For details see the documentation of the various devices types in *Interfaces* . Good practice is to query the current parameters first, and to modify only those parameters not suitable for the application. When the application calls the *VIDIOC_S_FMT* ioctl with a pointer to a struct *v4l2_format* structure the driver checks and adjusts the parameters against hardware abilities. Drivers should not return an error code unless the type field is invalid, this is a mechanism to fathom device capabilities and to approach parameters acceptable for both the application and driver. On success the driver may program the hardware, allocate resources and generally prepare for data exchange. Finally the *VIDIOC_S_FMT* ioctl returns the current format parameters as *VIDIOC_G_FMT* does. Very simple, inflexible devices may even ignore all input and always return the default parameters. However all V4L2 devices exchanging data with the application must implement the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl. When the requested buffer type is not supported drivers return an EINVAL error code on a *VIDIOC_S_FMT* attempt. When I/O is already in progress or the resource is not available for other reasons drivers return the EBUSY error code.

The *VIDIOC_TRY_FMT* ioctl is equivalent to *VIDIOC_S_FMT* with one exception: it does not change driver state. It can also be called at any time, never returning EBUSY. This function is provided to negotiate parameters, to learn about hardware limitations, without disabling I/O or possibly time consuming hardware preparations. Although strongly recommended drivers are not required to implement this ioctl.

The format as returned by *VIDIOC_TRY_FMT* must be identical to what *VIDIOC_S_FMT* returns for the same input or output.

v4l2_format

Table 1.119: struct v4l2_format

__u32	type		Type of the data stream, see *v4l2_buf_type*.
union	fmt		
	struct *v4l2_pix_format*	pix	Definition of an image format, see *Image Formats*, used by video capture and output devices.
	struct *v4l2_pix_format_mplane*	pix_mp	Definition of an image format, see *Image Formats*, used by video capture and output devices that support the *multi-planar version of the API*.
	struct *v4l2_window*	win	Definition of an overlaid image, see *Video Overlay Interface*, used by video overlay devices.
	struct *v4l2_vbi_format*	vbi	Raw VBI capture or output parameters. This is discussed in more detail in *Raw VBI Data Interface*. Used by raw VBI capture and output devices.
	struct *v4l2_sliced_vbi_format*	sliced	Sliced VBI capture or output parameters. See *Sliced VBI Data Interface* for details. Used by sliced VBI capture and output devices.
	struct *v4l2_sdr_format*	sdr	Definition of a data format, see *Image Formats*, used by SDR capture and output devices.
	__u8	raw_data[200]	Place holder for future extensions.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_format* `type` field is invalid or the requested buffer type not supported.

ioctl VIDIOC_G_FREQUENCY, VIDIOC_S_FREQUENCY

Name

VIDIOC_G_FREQUENCY - VIDIOC_S_FREQUENCY - Get or set tuner or modulator radio frequency

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_FREQUENCY*, struct *v4l2_frequency* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_FREQUENCY*, const struct *v4l2_frequency* *argp)

Arguments

fd File descriptor returned by *open()*.

`argp`

Description

To get the current tuner or modulator radio frequency applications set the `tuner` field of a struct *v4l2_frequency* to the respective tuner or modulator number (only input devices have tuners, only output devices have modulators), zero out the `reserved` array and call the *VIDIOC_G_FREQUENCY* ioctl with a pointer to this structure. The driver stores the current frequency in the `frequency` field.

To change the current tuner or modulator radio frequency applications initialize the tuner, type and frequency fields, and the reserved array of a struct *v4l2_frequency* and call the *VIDIOC_S_FREQUENCY* ioctl with a pointer to this structure. When the requested frequency is not possible the driver assumes the closest possible value. However *VIDIOC_S_FREQUENCY* is a write-only ioctl, it does not return the actual new frequency.

v4l2_frequency

Table 1.120: struct v4l2_frequency

__u32	tuner	The tuner or modulator index number. This is the same value as in the struct *v4l2_input* tuner field and the struct *v4l2_tuner* index field, or the struct *v4l2_output* modulator field and the struct *v4l2_modulator* index field.
__u32	type	The tuner type. This is the same value as in the struct *v4l2_tuner* type field. The type must be set to V4L2_TUNER_RADIO for /dev/radioX device nodes, and to V4L2_TUNER_ANALOG_TV for all others. Set this field to V4L2_TUNER_RADIO for modulators (currently only radio modulators are supported). See *v4l2_tuner_type*
__u32	frequency	Tuning frequency in units of 62.5 kHz, or if the struct *v4l2_tuner* or struct *v4l2_modulator* capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz. A 1 Hz unit is used when the capability flag V4L2_TUNER_CAP_1HZ is set.
__u32	reserved[8]	Reserved for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The tuner index is out of bounds or the value in the type field is wrong.

EBUSY A hardware seek is in progress.

ioctl VIDIOC_G_INPUT, VIDIOC_S_INPUT

Name

VIDIOC_G_INPUT - VIDIOC_S_INPUT - Query or select the current video input

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_INPUT*, int **argp*)

int **ioctl**(int *fd*, *VIDIOC_S_INPUT*, int **argp*)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the current video input applications call the *VIDIOC_G_INPUT* ioctl with a pointer to an integer where the driver stores the number of the input, as in the struct *v4l2_input* index field. This ioctl will fail only when there are no video inputs, returning EINVAL.

To select a video input applications store the number of the desired input in an integer and call the *VIDIOC_S_INPUT* ioctl with a pointer to this integer. Side effects are possible. For example inputs may support different video standards, so the driver may implicitly switch the current standard. Because of these possible side effects applications must select an input before querying or negotiating any other parameters.

Information about video inputs is available using the *ioctl VIDIOC_ENUMINPUT* ioctl.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The number of the video input is out of bounds.

ioctl VIDIOC_G_JPEGCOMP, VIDIOC_S_JPEGCOMP

Name

VIDIOC_G_JPEGCOMP - VIDIOC_S_JPEGCOMP

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_JPEGCOMP*, *v4l2_jpegcompression* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_JPEGCOMP*, const *v4l2_jpegcompression* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

These ioctls are **deprecated**. New drivers and applications should use *JPEG class controls* for image quality and JPEG markers control.

[to do]

Ronald Bultje elaborates:

APP is some application-specific information. The application can set it itself, and it'll be stored in the JPEG-encoded fields (eg; interlacing information for in an AVI or so). COM is the same, but it's comments, like 'encoded by me' or so.

jpeg_markers describes whether the huffman tables, quantization tables and the restart interval information (all JPEG-specific stuff) should be stored in the JPEG-encoded fields. These define how the JPEG field is encoded. If you omit them, applications assume you've used standard encoding. You usually do want to add them.

v4l2_jpegcompression

Table 1.121: struct v4l2_jpegcompression

int	quality	Deprecated. If *V4L2_CID_JPEG_COMPRESSION_QUALITY* control is exposed by a driver applications should use it instead and ignore this field.
int	APPn	
int	APP_len	
char	APP_data[60]	
int	COM_len	
char	COM_data[60]	
__u32	jpeg_markers	See *JPEG Markers Flags* . Deprecated. If *V4L2_CID_JPEG_ACTIVE_MARKER* control is exposed by a driver applications should use it instead and ignore this field.

Table 1.122: JPEG Markers Flags

V4L2_JPEG_MARKER_DHT	(1<<3)	Define Huffman Tables
V4L2_JPEG_MARKER_DQT	(1<<4)	Define Quantization Tables
V4L2_JPEG_MARKER_DRI	(1<<5)	Define Restart Interval
V4L2_JPEG_MARKER_COM	(1<<6)	Comment segment
V4L2_JPEG_MARKER_APP	(1<<7)	App segment, driver will always use APP0

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_G_MODULATOR, VIDIOC_S_MODULATOR

Name

VIDIOC_G_MODULATOR - VIDIOC_S_MODULATOR - Get or set modulator attributes

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_MODULATOR*, struct *v4l2_modulator* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_MODULATOR*, const struct *v4l2_modulator* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the attributes of a modulator applications initialize the index field and zero out the reserved array of a struct *v4l2_modulator* and call the *VIDIOC_G_MODULATOR* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code when the index is out of bounds. To enumerate all modulators applications shall begin at index zero, incrementing by one until the driver returns EINVAL.

Modulators have two writable properties, an audio modulation set and the radio frequency. To change the modulated audio subprograms, applications initialize the index and txsubchans fields and the reserved

array and call the *VIDIOC_S_MODULATOR* ioctl. Drivers may choose a different audio modulation if the request cannot be satisfied. However this is a write-only ioctl, it does not return the actual audio modulation selected.

SDR specific modulator types are V4L2_TUNER_SDR and V4L2_TUNER_RF. For SDR devices txsubchans field must be initialized to zero. The term 'modulator' means SDR transmitter in this context.

To change the radio frequency the *VIDIOC_S_FREQUENCY* ioctl is available.

v4l2_modulator

Table 1.123: struct v4l2_modulator

__u32	index	Identifies the modulator, set by the application.
__u8	name[32]	Name of the modulator, a NUL-terminated ASCII string. This information is intended for the user.
__u32	capability	Modulator capability flags. No flags are defined for this field, the tuner flags in struct *v4l2_tuner* are used accordingly. The audio flags indicate the ability to encode audio subprograms. They will *not* change for example with the current video standard.
__u32	rangelow	The lowest tunable frequency in units of 62.5 KHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz.
__u32	rangehigh	The highest tunable frequency in units of 62.5 KHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz.
__u32	txsubchans	With this field applications can determine how audio sub-carriers shall be modulated. It contains a set of flags as defined in *Modulator Audio Transmission Flags* . ***Note:*** *The tuner rxsubchans flags are reused, but the semantics are different. Video output devices are assumed to have an analog or PCM audio input with 1-3 channels. The txsubchans flags select one or more channels for modulation, together with some audio subprogram indicator, for example, a stereo pilot tone.*
__u32	type	Type of the modulator, see *v4l2_tuner_type*.
__u32	reserved[3]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 1.124: Modulator Audio Transmission Flags

V4L2_TUNER_SUB_MONO	0x0001	Modulate channel 1 as mono audio, when the input has more channels, a down-mix of channel 1 and 2. This flag does not combine with V4L2_TUNER_SUB_STEREO or V4L2_TUNER_SUB_LANG1.
V4L2_TUNER_SUB_STEREO	0x0002	Modulate channel 1 and 2 as left and right channel of a stereo audio signal. When the input has only one channel or two channels and V4L2_TUNER_SUB_SAP is also set, channel 1 is encoded as left and right channel. This flag does not combine with V4L2_TUNER_SUB_MONO or V4L2_TUNER_SUB_LANG1. When the driver does not support stereo audio it shall fall back to mono.
V4L2_TUNER_SUB_LANG1	0x0008	Modulate channel 1 and 2 as primary and secondary language of a bilingual audio signal. When the input has only one channel it is used for both languages. It is not possible to encode the primary or secondary language only. This flag does not combine with V4L2_TUNER_SUB_MONO, V4L2_TUNER_SUB_STEREO or V4L2_TUNER_SUB_SAP. If the hardware does not support the respective audio matrix, or the current video standard does not permit bilingual audio the *VIDIOC_S_MODULATOR* ioctl shall return an EINVAL error code and the driver shall fall back to mono or stereo mode.
V4L2_TUNER_SUB_LANG2	0x0004	Same effect as V4L2_TUNER_SUB_SAP.
V4L2_TUNER_SUB_SAP	0x0004	When combined with V4L2_TUNER_SUB_MONO the first channel is encoded as mono audio, the last channel as Second Audio Program. When the input has only one channel it is used for both audio tracks. When the input has three channels the mono track is a down-mix of channel 1 and 2. When combined with V4L2_TUNER_SUB_STEREO channel 1 and 2 are encoded as left and right stereo audio, channel 3 as Second Audio Program. When the input has only two channels, the first is encoded as left and right channel and the second as SAP. When the input has only one channel it is used for all audio tracks. It is not possible to encode a Second Audio Program only. This flag must combine with V4L2_TUNER_SUB_MONO or V4L2_TUNER_SUB_STEREO. If the hardware does not support the respective audio matrix, or the current video standard does not permit SAP the *VIDIOC_S_MODULATOR* ioctl shall return an EINVAL error code and driver shall fall back to mono or stereo mode.
V4L2_TUNER_SUB_RDS	0x0010	Enable the RDS encoder for a radio FM transmitter.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_modulator* index is out of bounds.

ioctl VIDIOC_G_OUTPUT, VIDIOC_S_OUTPUT

Name

VIDIOC_G_OUTPUT - VIDIOC_S_OUTPUT - Query or select the current video output

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_OUTPUT*, int *argp)

int **ioctl**(int *fd*, *VIDIOC_S_OUTPUT*, int *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the current video output applications call the *VIDIOC_G_OUTPUT* ioctl with a pointer to an integer where the driver stores the number of the output, as in the struct *v4l2_output* index field. This ioctl will fail only when there are no video outputs, returning the EINVAL error code.

To select a video output applications store the number of the desired output in an integer and call the *VIDIOC_S_OUTPUT* ioctl with a pointer to this integer. Side effects are possible. For example outputs may support different video standards, so the driver may implicitly switch the current standard. standard. Because of these possible side effects applications must select an output before querying or negotiating any other parameters.

Information about video outputs is available using the *ioctl VIDIOC_ENUMOUTPUT* ioctl.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The number of the video output is out of bounds, or there are no video outputs at all.

ioctl VIDIOC_G_PARM, VIDIOC_S_PARM

Name

VIDIOC_G_PARM - VIDIOC_S_PARM - Get or set streaming parameters

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_PARM*, *v4l2_streamparm* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_PARM*, *v4l2_streamparm* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

The current video standard determines a nominal number of frames per second. If less than this number of frames is to be captured or output, applications can request frame skipping or duplicating on the driver side. This is especially useful when using the *read()* or *write()* , which are not augmented by timestamps or sequence counters, and to avoid unnecessary data copying.

Further these ioctls can be used to determine the number of buffers used internally by a driver in read/write mode. For implications see the section discussing the *read()* function.

To get and set the streaming parameters applications call the *VIDIOC_G_PARM* and *VIDIOC_S_PARM* ioctl, respectively. They take a pointer to a struct *v4l2_streamparm* which contains a union holding separate parameters for input and output devices.

v4l2_streamparm

Table 1.125: struct v4l2_streamparm

__u32	type		The buffer (stream) type, same as struct *v4l2_format* type, set by the application. See *v4l2_buf_type*
union	parm		
	struct *v4l2_captureparm*	capture	Parameters for capture devices, used when type is V4L2_BUF_TYPE_VIDEO_CAPTURE.
	struct *v4l2_outputparm*	output	Parameters for output devices, used when type is V4L2_BUF_TYPE_VIDEO_OUTPUT.
	__u8	raw_data[200]	A place holder for future extensions.

v4l2_captureparm

Table 1.126: struct v4l2_captureparm

__u32	capability	See *Streaming Parameters Capabilites* .
__u32	capturemode	Set by drivers and applications, see *Capture Parameters Flags* .
struct *v4l2_fract*	timeperframe	This is the desired period between successive frames captured by the driver, in seconds. The field is intended to skip frames on the driver side, saving I/O bandwidth. Applications store here the desired frame period, drivers return the actual frame period, which must be greater or equal to the nominal frame period determined by the current video standard (struct *v4l2_standard* frameperiod field). Changing the video standard (also implicitly by switching the video input) may reset this parameter to the nominal frame period. To reset manually applications can just set this field to zero. Drivers support this function only when they set the V4L2_CAP_TIMEPERFRAME flag in the capability field.
__u32	extendedmode	Custom (driver specific) streaming parameters. When unused, applications and drivers must set this field to zero. Applications using this field should check the driver name and version, see *Querying Capabilities* .
__u32	readbuffers	Applications set this field to the desired number of buffers used internally by the driver in *read()* mode. Drivers return the actual number of buffers. When an application requests zero buffers, drivers should just return the current setting rather than the minimum or an error code. For details see *Read/Write* .
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set the array to zero.

v4l2_outputparm

Table 1.127: struct v4l2_outputparm

__u32	capability	See *Streaming Parameters Capabilites* .
__u32	outputmode	Set by drivers and applications, see *Capture Parameters Flags* .
struct *v4l2_fract*	timeperframe	This is the desired period between successive frames output by the driver, in seconds.
The field is intended to repeat frames on the driver side in *write()* mode (in streaming mode timestamps can be used to throttle the output), saving I/O bandwidth. Applications store here the desired frame period, drivers return the actual frame period, which must be greater or equal to the nominal frame period determined by the current video standard (struct *v4l2_standard* frameperiod field). Changing the video standard (also implicitly by switching the video output) may reset this parameter to the nominal frame period. To reset manually applications can just set this field to zero. Drivers support this function only when they set the V4L2_CAP_TIMEPERFRAME flag in the capability field.		
__u32	extendedmode	Custom (driver specific) streaming parameters. When unused, applications and drivers must set this field to zero. Applications using this field should check the driver name and version, see *Querying Capabilities* .
__u32	writebuffers	Applications set this field to the desired number of buffers used internally by the driver in *write()* mode. Drivers return the actual number of buffers. When an application requests zero buffers, drivers should just return the current setting rather than the minimum or an error code. For details see *Read/Write* .
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 1.128: Streaming Parameters Capabilites

V4L2_CAP_TIMEPERFRAME	0x1000	The frame skipping/repeating controlled by the timeperframe field is supported.

Table 1.129: Capture Parameters Flags

V4L2_MODE_HIGHQUALITY	0x0001	High quality imaging mode. High quality mode is intended for still imaging applications. The idea is to get the best possible image quality that the hardware can deliver. It is not defined how the driver writer may achieve that; it will depend on the hardware and the ingenuity of the driver writer. High quality mode is a different mode from the regular motion video capture modes. In high quality mode: • The driver may be able to capture higher resolutions than for motion capture. • The driver may support fewer pixel formats than motion capture (eg; true color). • The driver may capture and arithmetically combine multiple successive fields or frames to remove color edge artifacts and reduce the noise in the video data. • The driver may capture images in slices like a scanner in order to handle larger format images than would otherwise be possible. • An image capture operation may be significantly slower than motion capture. • Moving objects in the image might have excessive motion blur. • Capture might only work through the *read()* call.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_G_PRIORITY, VIDIOC_S_PRIORITY

Name

VIDIOC_G_PRIORITY - VIDIOC_S_PRIORITY - Query or request the access priority associated with a file descriptor

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_PRIORITY*, enum *v4l2_priority* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_PRIORITY*, const enum *v4l2_priority* *argp)

Arguments

fd File descriptor returned by *open()* .

argp Pointer to an enum v4l2_priority type.

Description

To query the current access priority applications call the *VIDIOC_G_PRIORITY* ioctl with a pointer to an enum v4l2_priority variable where the driver stores the current priority.

To request an access priority applications store the desired priority in an enum v4l2_priority variable and call *VIDIOC_S_PRIORITY* ioctl with a pointer to this variable.

v4l2_priority

Table 1.130: enum v4l2_priority

V4L2_PRIORITY_UNSET	0	
V4L2_PRIORITY_BACKGROUND	1	Lowest priority, usually applications running in background, for example monitoring VBI transmissions. A proxy application running in user space will be necessary if multiple applications want to read from a device at this priority.
V4L2_PRIORITY_INTERACTIVE	2	
V4L2_PRIORITY_DEFAULT	2	Medium priority, usually applications started and interactively controlled by the user. For example TV viewers, Teletext browsers, or just "panel" applications to change the channel or video controls. This is the default priority unless an application requests another.
V4L2_PRIORITY_RECORD	3	Highest priority. Only one file descriptor can have this priority, it blocks any other fd from changing device properties. Usually applications which must not be interrupted, like video recording.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The requested priority value is invalid.

EBUSY Another application already requested higher priority.

ioctl VIDIOC_G_SELECTION, VIDIOC_S_SELECTION

Name

VIDIOC_G_SELECTION - VIDIOC_S_SELECTION - Get or set one of the selection rectangles

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_SELECTION*, struct *v4l2_selection* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_SELECTION*, struct *v4l2_selection* *argp)

Arguments

fd File descriptor returned by *open()* .

request VIDIOC_G_SELECTION, VIDIOC_S_SELECTION

argp

Description

The ioctls are used to query and configure selection rectangles.

To query the cropping (composing) rectangle set struct *v4l2_selection* type field to the respective buffer type. Do not use the multiplanar buffer types. Use V4L2_BUF_TYPE_VIDEO_CAPTURE instead of V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE and use V4L2_BUF_TYPE_VIDEO_OUTPUT instead of V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE. The next step is setting the value of struct *v4l2_selection* target field to V4L2_SEL_TGT_CROP (V4L2_SEL_TGT_COMPOSE). Please refer to table *Common selection definitions* or *API for cropping, composing and scaling* for additional targets. The flags and reserved fields of struct *v4l2_selection* are ignored and they must be filled with zeros. The driver fills the rest of the structure or returns EINVAL error code if incorrect buffer type or target was used. If cropping (composing) is not supported then the active rectangle is not mutable and it is always equal to the bounds rectangle. Finally, the struct *v4l2_rect* r rectangle is filled with the current cropping (composing) coordinates. The coordinates are expressed in driver-dependent units. The only exception are rectangles for images in raw formats, whose coordinates are always expressed in pixels.

To change the cropping (composing) rectangle set the struct *v4l2_selection* type field to the respective buffer type. Do not use multiplanar buffers. Use V4L2_BUF_TYPE_VIDEO_CAPTURE instead of V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE. Use V4L2_BUF_TYPE_VIDEO_OUTPUT instead of V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE. The next step is setting the value of struct *v4l2_selection* target to V4L2_SEL_TGT_CROP (V4L2_SEL_TGT_COMPOSE). Please refer to table *Common selection definitions* or *API for cropping, composing and scaling* for additional targets. The struct *v4l2_rect* r rectangle need to be set to the desired active area. Field struct *v4l2_selection* reserved is ignored and must be filled with zeros. The driver may adjust coordinates of the requested rectangle. An application may introduce constraints to control rounding behaviour. The struct *v4l2_selection* flags field must be set to one of the following:

- 0 - The driver can adjust the rectangle size freely and shall choose a crop/compose rectangle as close as possible to the requested one.
- V4L2_SEL_FLAG_GE - The driver is not allowed to shrink the rectangle. The original rectangle must lay inside the adjusted one.
- V4L2_SEL_FLAG_LE - The driver is not allowed to enlarge the rectangle. The adjusted rectangle must lay inside the original one.
- V4L2_SEL_FLAG_GE | V4L2_SEL_FLAG_LE - The driver must choose the size exactly the same as in the requested rectangle.

Please refer to *Size adjustments with constraint flags.* .

The driver may have to adjusts the requested dimensions against hardware limits and other parts as the pipeline, i.e. the bounds given by the capture/output window or TV display. The closest possible values of horizontal and vertical offset and sizes are chosen according to following priority:

1. Satisfy constraints from struct *v4l2_selection* flags.
2. Adjust width, height, left, and top to hardware limits and alignments.
3. Keep center of adjusted rectangle as close as possible to the original one.
4. Keep width and height as close as possible to original ones.
5. Keep horizontal and vertical offset as close as possible to original ones.

On success the struct *v4l2_rect* r field contains the adjusted rectangle. When the parameters are unsuitable the application may modify the cropping (composing) or image parameters and repeat the cycle until satisfactory parameters have been negotiated. If constraints flags have to be violated at then ERANGE is returned. The error indicates that *there exist no rectangle* that satisfies the constraints.

Selection targets and flags are documented in *Common selection definitions* .

Fig. 1.14: Size adjustments with constraint flags.
Behaviour of rectangle adjustment for different constraint flags.

v4l2_selection

Table 1.131: struct v4l2_selection

__u32	type	Type of the buffer (from enum *v4l2_buf_type*).
__u32	target	Used to select between *cropping and composing rectangles* .
__u32	flags	Flags controlling the selection rectangle adjustments, refer to *selection flags* .
struct *v4l2_rect*	r	The selection rectangle.
__u32	reserved[9]	Reserved fields for future use. Drivers and applications must zero this array.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL Given buffer type type or the selection target target is not supported, or the flags argument is not valid.

ERANGE It is not possible to adjust struct *v4l2_rect* r rectangle to satisfy all constraints given in the flags argument.

ENODATA Selection is not supported for this input or output.

EBUSY It is not possible to apply change of the selection rectangle at the moment. Usually because streaming is in progress.

ioctl VIDIOC_G_SLICED_VBI_CAP

Name

VIDIOC_G_SLICED_VBI_CAP - Query sliced VBI capabilities

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_SLICED_VBI_CAP*, struct *v4l2_sliced_vbi_cap* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To find out which data services are supported by a sliced VBI capture or output device, applications initialize the type field of a struct *v4l2_sliced_vbi_cap*, clear the reserved array and call the *VIDIOC_G_SLICED_VBI_CAP* ioctl. The driver fills in the remaining fields or returns an EINVAL error code if the sliced VBI API is unsupported or type is invalid.

Note:

The type field was added, and the ioctl changed from read-only to write-read, in Linux 2.6.19.

v4l2_sliced_vbi_cap

Table 1.132: struct v4l2_sliced_vbi_cap

__u16	service_set	A set of all data services supported by the driver. Equal to the union of all elements of the service_lines array.		
__u16	service_lines[2][24]	Each element of this array contains a set of data services the hardware can look for or insert into a particular scan line. Data services are defined in *Sliced VBI services* . Array indices map to ITU-R line numbers[1] as follows:		
		Element	525 line systems	625 line systems
		service_lines[0][1]	1	1
		service_lines[0][23]	23	23
		service_lines[1][1]	264	314
		service_lines[1][23]	286	336
		The number of VBI lines the hardware can capture or output per frame, or the number of services it can identify on a given line may be limited. For example on PAL line 16 the hardware may be able to look for a VPS or Teletext signal, but not both at the same time. Applications can learn about these limits using the *VIDIOC_S_FMT* ioctl as described in *Sliced VBI Data Interface* .		
		Drivers must set service_lines [0][0] and service_lines[1][0] to zero.		
__u32	type	Type of the data stream, see *v4l2_buf_type*. Should be V4L2_BUF_TYPE_SLICED_VBI_CAPTURE or V4L2_BUF_TYPE_SLICED_VBI_OUTPUT.		
__u32	reserved[3]	This array is reserved for future extensions. Applications and drivers must set it to zero.		

[1] See also *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering* .

Table 1.133: Sliced VBI services

Symbol	Value	Reference	Lines, usually	Payload
V4L2_SLICED_TELETEXT_B (Teletext System B)	0x0001	*ETS 300 706* , *ITU BT.653*	PAL/SECAM line 7-22, 320-335 (second field 7-22)	Last 42 of the 45 byte Teletext packet, that is without clock run-in and framing code, lsb first transmitted.
V4L2_SLICED_VPS	0x0400	*ETS 300 231*	PAL line 16	Byte number 3 to 15 according to Figure 9 of ETS 300 231, lsb first transmitted.
V4L2_SLICED_CAPTION_525	0x1000	*CEA 608-E*	NTSC line 21, 284 (second field 21)	Two bytes in transmission order, including parity bit, lsb first transmitted.
V4L2_SLICED_WSS_625	0x4000	*EN 300 294* , *ITU BT.1119*	PAL/SECAM line 23	Byte 0 1 msb lsb msb lsb Bit 7 6 5 4 3 2 1 0 x x 13 12 11 10 9
V4L2_SLICED_VBI_525	0x1000	Set of services applicable to 525 line systems.		
V4L2_SLICED_VBI_625	0x4401	Set of services applicable to 625 line systems.		

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The value in the `type` field is wrong.

ioctl VIDIOC_G_STD, VIDIOC_S_STD

Name

VIDIOC_G_STD - VIDIOC_S_STD - Query or select the video standard of the current input

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_STD*, v4l2_std_id **argp*)

int **ioctl**(int *fd*, *VIDIOC_S_STD*, const v4l2_std_id **argp*)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

To query and select the current video standard applications use the *VIDIOC_G_STD* and *VIDIOC_S_STD* ioctls which take a pointer to a *v4l2_std_id* type as argument. *VIDIOC_G_STD* can return a single flag or a set of flags as in struct *v4l2_standard* field `id`. The flags must be unambiguous such that they appear in only one enumerated struct *v4l2_standard* structure.

VIDIOC_S_STD accepts one or more flags, being a write-only ioctl it does not return the actual new standard as *VIDIOC_G_STD* does. When no flags are given or the current input does not support the requested standard the driver returns an `EINVAL` error code. When the standard set is ambiguous drivers may return `EINVAL` or choose any of the requested standards. If the current input or output does not

support standard video timings (e.g. if *ioctl VIDIOC_ENUMINPUT* does not set the V4L2_IN_CAP_STD flag), then ENODATA error code is returned.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The *VIDIOC_S_STD* parameter was unsuitable.

ENODATA Standard video timings are not supported for this input or output.

ioctl VIDIOC_G_TUNER, VIDIOC_S_TUNER

Name

VIDIOC_G_TUNER - VIDIOC_S_TUNER - Get or set tuner attributes

Synopsis

int **ioctl**(int *fd*, *VIDIOC_G_TUNER*, struct *v4l2_tuner* *argp)

int **ioctl**(int *fd*, *VIDIOC_S_TUNER*, const struct *v4l2_tuner* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the attributes of a tuner applications initialize the index field and zero out the reserved array of a struct *v4l2_tuner* and call the VIDIOC_G_TUNER ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code when the index is out of bounds. To enumerate all tuners applications shall begin at index zero, incrementing by one until the driver returns EINVAL.

Tuners have two writable properties, the audio mode and the radio frequency. To change the audio mode, applications initialize the index, audmode and reserved fields and call the VIDIOC_S_TUNER ioctl. This will *not* change the current tuner, which is determined by the current video input. Drivers may choose a different audio mode if the requested mode is invalid or unsupported. Since this is a write-only ioctl, it does not return the actually selected audio mode.

SDR specific tuner types are V4L2_TUNER_SDR and V4L2_TUNER_RF. For SDR devices audmode field must be initialized to zero. The term 'tuner' means SDR receiver in this context.

To change the radio frequency the *VIDIOC_S_FREQUENCY* ioctl is available.

v4l2_tuner

Table 1.134: struct v4l2_tuner

__u32	index	Identifies the tuner, set by the application.

Continued on next page

Table 1.134 – continued from previous page

__u8	name[32]	Name of the tuner, a NUL-terminated ASCII string. This information is intended for the user.	
__u32	type	Type of the tuner, see *v4l2_tuner_type*.	
__u32	capability	Tuner capability flags, see *Tuner and Modulator Capability Flags* . Audio flags indicate the ability to decode audio subprograms. They will *not* change, for example with the current video standard. When the structure refers to a radio tuner the V4L2_TUNER_CAP_LANG1, V4L2_TUNER_CAP_LANG2 and V4L2_TUNER_CAP_NORM flags can't be used. If multiple frequency bands are supported, then capability is the union of all capability fields of each struct *v4l2_frequency_band*.	
__u32	rangelow	The lowest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz. If multiple frequency bands are supported, then rangelow is the lowest frequency of all the frequency bands.	
__u32	rangehigh	The highest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz. If multiple frequency bands are supported, then rangehigh is the highest frequency of all the frequency bands.	
__u32	rxsubchans	Some tuners or audio decoders can determine the received audio subprograms by analyzing audio carriers, pilot tones or other indicators. To pass this information drivers set flags defined in *Tuner Audio Reception Flags* in this field. For example:	
		V4L2_TUNER_SUB_MONO	receiving mono audio
		STEREO | SAP	receiving stereo audio and a secondary audio program
		MONO | STEREO	receiving mono or stereo audio, the hardware cannot distinguish
		LANG1 | LANG2	receiving bilingual audio
		MONO | STEREO | LANG1 | LANG2	receiving mono, stereo or bilingual audio
		When the V4L2_TUNER_CAP_STEREO, _LANG1, _LANG2 or _SAP flag is cleared in the capability field, the corresponding V4L2_TUNER_SUB_ flag must not be set here. This field is valid only if this is the tuner of the current video input, or when the structure refers to a radio tuner.	

Continued on next page

Table 1.134 – continued from previous page

__u32	audmode	The selected audio mode, see *Tuner Audio Modes* for valid values. The audio mode does not affect audio subprogram detection, and like a *User Controls* it does not automatically change unless the requested mode is invalid or unsupported. See *Tuner Audio Matrix* for possible results when the selected and received audio programs do not match. Currently this is the only field of struct struct *v4l2_tuner* applications can change.
__u32	signal	The signal strength if known. Ranging from 0 to 65535. Higher values indicate a better signal.
__s32	afc	Automatic frequency control. When the afc value is negative, the frequency is too low, when positive too high.
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set the array to zero.

v4l2_tuner_type

Table 1.135: enum v4l2_tuner_type

V4L2_TUNER_RADIO	1	Tuner supports radio
V4L2_TUNER_ANALOG_TV	2	Tuner supports analog TV
V4L2_TUNER_SDR	4	Tuner controls the A/D and/or D/A block of a Software Digital Radio (SDR)
V4L2_TUNER_RF	5	Tuner controls the RF part of a Software Digital Radio (SDR)

Table 1.136: Tuner and Modulator Capability Flags

V4L2_TUNER_CAP_LOW	0x0001	When set, tuning frequencies are expressed in units of 62.5 Hz instead of 62.5 kHz.
V4L2_TUNER_CAP_NORM	0x0002	This is a multi-standard tuner; the video standard can or must be switched. (B/G PAL tuners for example are typically not considered multi-standard because the video standard is automatically determined from the frequency band.) The set of supported video standards is available from the struct *v4l2_input* pointing to this tuner, see the description of ioctl *ioctl VIDIOC_ENUMINPUT* for details. Only V4L2_TUNER_ANALOG_TV tuners can have this capability.
V4L2_TUNER_CAP_HWSEEK_BOUNDED	0x0004	If set, then this tuner supports the hardware seek functionality where the seek stops when it reaches the end of the frequency range.
V4L2_TUNER_CAP_HWSEEK_WRAP	0x0008	If set, then this tuner supports the hardware seek functionality where the seek wraps around when it reaches the end of the frequency range.

Continued on next page

Table 1.136 – continued from previous page

V4L2_TUNER_CAP_STEREO	0x0010	Stereo audio reception is supported.
V4L2_TUNER_CAP_LANG1	0x0040	Reception of the primary language of a bilingual audio program is supported. Bilingual audio is a feature of two-channel systems, transmitting the primary language monaural on the main audio carrier and a secondary language monaural on a second carrier. Only V4L2_TUNER_ANALOG_TV tuners can have this capability.
V4L2_TUNER_CAP_LANG2	0x0020	Reception of the secondary language of a bilingual audio program is supported. Only V4L2_TUNER_ANALOG_TV tuners can have this capability.
V4L2_TUNER_CAP_SAP	0x0020	Reception of a secondary audio program is supported. This is a feature of the BTSC system which accompanies the NTSC video standard. Two audio carriers are available for mono or stereo transmissions of a primary language, and an independent third carrier for a monaural secondary language. Only V4L2_TUNER_ANALOG_TV tuners can have this capability. ***Note:*** *The V4L2_TUNER_CAP_LANG2 and V4L2_TUNER_CAP_SAP flags are synonyms. V4L2_TUNER_CAP_SAP applies when the tuner supports the V4L2_STD_NTSC_M video standard.*
V4L2_TUNER_CAP_RDS	0x0080	RDS capture is supported. This capability is only valid for radio tuners.
V4L2_TUNER_CAP_RDS_BLOCK_IO	0x0100	The RDS data is passed as unparsed RDS blocks.
V4L2_TUNER_CAP_RDS_CONTROLS	0x0200	The RDS data is parsed by the hardware and set via controls.
V4L2_TUNER_CAP_FREQ_BANDS	0x0400	The *ioctl VIDIOC_ENUM_FREQ_BANDS* ioctl can be used to enumerate the available frequency bands.
V4L2_TUNER_CAP_HWSEEK_PROG_LIM	0x0800	The range to search when using the hardware seek functionality is programmable, see *ioctl VIDIOC_S_HW_FREQ_SEEK* for details.
V4L2_TUNER_CAP_1HZ	0x1000	When set, tuning frequencies are expressed in units of 1 Hz instead of 62.5 kHz.

Table 1.137: Tuner Audio Reception Flags

V4L2_TUNER_SUB_MONO	0x0001	The tuner receives a mono audio signal.
V4L2_TUNER_SUB_STEREO	0x0002	The tuner receives a stereo audio signal.
V4L2_TUNER_SUB_LANG1	0x0008	The tuner receives the primary language of a bilingual audio signal. Drivers must clear this flag when the current video standard is V4L2_STD_NTSC_M.
V4L2_TUNER_SUB_LANG2	0x0004	The tuner receives the secondary language of a bilingual audio signal (or a second audio program).
V4L2_TUNER_SUB_SAP	0x0004	The tuner receives a Second Audio Program. ***Note:*** *The V4L2_TUNER_SUB_LANG2 and V4L2_TUNER_SUB_SAP flags are synonyms. The V4L2_TUNER_SUB_SAP flag applies when the current video standard is V4L2_STD_NTSC_M.*
V4L2_TUNER_SUB_RDS	0x0010	The tuner receives an RDS channel.

Table 1.138: Tuner Audio Modes

V4L2_TUNER_MODE_MONO	0	Play mono audio. When the tuner receives a stereo signal this a down-mix of the left and right channel. When the tuner receives a bilingual or SAP signal this mode selects the primary language.
V4L2_TUNER_MODE_STEREO	1	Play stereo audio. When the tuner receives bilingual audio it may play different languages on the left and right channel or the primary language is played on both channels. Playing different languages in this mode is deprecated. New drivers should do this only in MODE_LANG1_LANG2. When the tuner receives no stereo signal or does not support stereo reception the driver shall fall back to MODE_MONO.
V4L2_TUNER_MODE_LANG1	3	Play the primary language, mono or stereo. Only V4L2_TUNER_ANALOG_TV tuners support this mode.
V4L2_TUNER_MODE_LANG2	2	Play the secondary language, mono. When the tuner receives no bilingual audio or SAP, or their reception is not supported the driver shall fall back to mono or stereo mode. Only V4L2_TUNER_ANALOG_TV tuners support this mode.
V4L2_TUNER_MODE_SAP	2	Play the Second Audio Program. When the tuner receives no bilingual audio or SAP, or their reception is not supported the driver shall fall back to mono or stereo mode. Only V4L2_TUNER_ANALOG_TV tuners support this mode. ***Note:*** *The V4L2_TUNER_MODE_LANG2 and V4L2_TUNER_MODE_SAP are synonyms.*
V4L2_TUNER_MODE_LANG1_LANG2	4	Play the primary language on the left channel, the secondary language on the right channel. When the tuner receives no bilingual audio or SAP, it shall fall back to MODE_LANG1 or MODE_MONO. Only V4L2_TUNER_ANALOG_TV tuners support this mode.

Table 1.139: Tuner Audio Matrix

	Selected V4L2_TUNER_MODE_				
Received V4L2_TUNER_SUB_	MONO	STEREO	LANG1	LANG2 = SAP	LANG1_LANG2 [1]
MONO	Mono	Mono/Mono	Mono	Mono	Mono/Mono
MONO | SAP	Mono	Mono/Mono	Mono	SAP	Mono/SAP (preferred) or Mono/Mono
STEREO	L+R	L/R	Stereo L/R (preferred) or Mono L+R	Stereo L/R (preferred) or Mono L+R	L/R (preferred) or L+R/L+R
STEREO | SAP	L+R	L/R	Stereo L/R (preferred) or Mono L+R	SAP	L+R/SAP (preferred) or L/R or L+R/L+R
LANG1 | LANG2	Language 1	Lang1/Lang2 (deprecated [2]) or Lang1/Lang1	Language 1	Language 2	Lang1/Lang2 (preferred) or Lang1/Lang1

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_tuner* `index` is out of bounds.

ioctl VIDIOC_LOG_STATUS

Name

VIDIOC_LOG_STATUS - Log driver status information

Synopsis

int **ioctl**(int *fd*, *VIDIOC_LOG_STATUS*)

Arguments

fd File descriptor returned by *open()* .

Description

As the video/audio devices become more complicated it becomes harder to debug problems. When this ioctl is called the driver will output the current device status to the kernel log. This is particular useful when dealing with problems like no sound, no video and incorrectly tuned channels. Also many modern devices autodetect video and audio standards and this ioctl will report what the device thinks what the standard is. Mismatches may give an indication where the problem is.

This ioctl is optional and not all drivers support it. It was introduced in Linux 2.6.15.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_OVERLAY

Name

VIDIOC_OVERLAY - Start or stop video overlay

Synopsis

int **ioctl**(int *fd*, *VIDIOC_OVERLAY*, const int *argp*)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

This ioctl is part of the *video overlay* I/O method. Applications call *ioctl VIDIOC_OVERLAY* to start or stop the overlay. It takes a pointer to an integer which must be set to zero by the application to stop overlay, to one to start.

Drivers do not support *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* or *VIDIOC_STREAMOFF* with `V4L2_BUF_TYPE_VIDEO_OVERLAY`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The overlay parameters have not been set up. See *Video Overlay Interface* for the necessary steps.

ioctl VIDIOC_PREPARE_BUF

Name

VIDIOC_PREPARE_BUF - Prepare a buffer for I/O

Synopsis

int **`ioctl`**(int *fd*, *VIDIOC_PREPARE_BUF*, struct *v4l2_buffer* *argp*)

Arguments

`fd` File descriptor returned by *open()* .

`argp`

Description

Applications can optionally call the *ioctl VIDIOC_PREPARE_BUF* ioctl to pass ownership of the buffer to the driver before actually enqueuing it, using the *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* ioctl, and to prepare it for future I/O. Such preparations may include cache invalidation or cleaning. Performing them in advance saves time during the actual I/O. In case such cache operations are not required, the application can use one of `V4L2_BUF_FLAG_NO_CACHE_INVALIDATE` and `V4L2_BUF_FLAG_NO_CACHE_CLEAN` flags to skip the respective step.

The struct *v4l2_buffer* structure is specified in *Buffers* .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY File I/O is in progress.

EINVAL The buffer `type` is not supported, or the `index` is out of bounds, or no buffers have been allocated yet, or the `userptr` or `length` are invalid.

ioctl VIDIOC_QBUF, VIDIOC_DQBUF

Name

VIDIOC_QBUF - VIDIOC_DQBUF - Exchange a buffer with the driver

Synopsis

int **ioctl**(int *fd*, *VIDIOC_QBUF*, struct *v4l2_buffer* *argp)

int **ioctl**(int *fd*, *VIDIOC_DQBUF*, struct *v4l2_buffer* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Applications call the VIDIOC_QBUF ioctl to enqueue an empty (capturing) or filled (output) buffer in the driver's incoming queue. The semantics depend on the selected I/O method.

To enqueue a buffer applications set the type field of a struct *v4l2_buffer* to the same buffer type as was previously used with struct *v4l2_format* type and struct *v4l2_requestbuffers* type. Applications must also set the index field. Valid index numbers range from zero to the number of buffers allocated with *ioctl VIDIOC_REQBUFS* (struct *v4l2_requestbuffers* count) minus one. The contents of the struct *v4l2_buffer* returned by a *ioctl VIDIOC_QUERYBUF* ioctl will do as well. When the buffer is intended for output (type is V4L2_BUF_TYPE_VIDEO_OUTPUT, V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE, or V4L2_BUF_TYPE_VBI_OUTPUT) applications must also initialize the bytesused, field and timestamp fields, see *Buffers* for details. Applications must also set flags to 0. The reserved2 and reserved fields must be set to 0. When using the *multi-planar API* , the m.planes field must contain a userspace pointer to a filled-in array of struct *v4l2_plane* and the length field must be set to the number of elements in that array.

To enqueue a *memory mapped* buffer applications set the memory field to V4L2_MEMORY_MMAP. When VIDIOC_QBUF is called with a pointer to this structure the driver sets the V4L2_BUF_FLAG_MAPPED and V4L2_BUF_FLAG_QUEUED flags and clears the V4L2_BUF_FLAG_DONE flag in the flags field, or it returns an EINVAL error code.

To enqueue a *user pointer* buffer applications set the memory field to V4L2_MEMORY_USERPTR, the m.userptr field to the address of the buffer and length to its size. When the multi-planar API is used, m.userptr and length members of the passed array of struct *v4l2_plane* have to be used instead. When VIDIOC_QBUF is called with a pointer to this structure the driver sets the V4L2_BUF_FLAG_QUEUED flag and clears the V4L2_BUF_FLAG_MAPPED and V4L2_BUF_FLAG_DONE flags in the flags field, or it returns an error code. This ioctl locks the memory pages of the buffer in physical memory, they cannot be swapped out to disk. Buffers remain locked until dequeued, until the *VIDIOC_STREAMOFF* or *ioctl VIDIOC_REQBUFS* ioctl is called, or until the device is closed.

To enqueue a *DMABUF* buffer applications set the memory field to V4L2_MEMORY_DMABUF and the m.fd field to a file descriptor associated with a DMABUF buffer. When the multi-planar API is used the m.fd fields of the passed array of struct *v4l2_plane* have to be used instead. When VIDIOC_QBUF is called with a pointer to this structure the driver sets the V4L2_BUF_FLAG_QUEUED flag and clears the V4L2_BUF_FLAG_MAPPED and V4L2_BUF_FLAG_DONE flags in the flags field, or it returns an error code. This ioctl locks the buffer. Locking a buffer means passing it to a driver for a hardware access (usually DMA). If an application accesses (reads/writes) a locked buffer then the result is undefined. Buffers remain locked until dequeued, until the *VIDIOC_STREAMOFF* or *ioctl VIDIOC_REQBUFS* ioctl is called, or until the device is closed.

Applications call the VIDIOC_DQBUF ioctl to dequeue a filled (capturing) or displayed (output) buffer from the driver's outgoing queue. They just set the type, memory and reserved fields of a struct *v4l2_buffer* as above, when VIDIOC_DQBUF is called with a pointer to this structure the driver fills the remaining fields or returns an error code. The driver may also set V4L2_BUF_FLAG_ERROR in the flags field. It indicates a non-critical (recoverable) streaming error. In such case the application may continue as normal, but should be aware that data in the dequeued buffer might be corrupted. When using the multi-planar API, the planes array must be passed in as well.

By default VIDIOC_DQBUF blocks when no buffer is in the outgoing queue. When the O_NONBLOCK flag was given to the *open()* function, VIDIOC_DQBUF returns immediately with an EAGAIN error code when no buffer is available.

The struct *v4l2_buffer* structure is specified in *Buffers* .

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EAGAIN Non-blocking I/O has been selected using O_NONBLOCK and no buffer was in the outgoing queue.

EINVAL The buffer type is not supported, or the index is out of bounds, or no buffers have been allocated yet, or the userptr or length are invalid.

EIO VIDIOC_DQBUF failed due to an internal error. Can also indicate temporary problems like signal loss.

> ***Note:***
>
> *The driver might dequeue an (empty) buffer despite returning an error, or even stop capturing. Reusing such buffer may be unsafe though and its details (e.g. index) may not be returned either. It is recommended that drivers indicate recoverable errors by setting the V4L2_BUF_FLAG_ERROR and returning 0 instead. In that case the application should be able to safely reuse the buffer and continue streaming.*

EPIPE VIDIOC_DQBUF returns this on an empty capture queue for mem2mem codecs if a buffer with the V4L2_BUF_FLAG_LAST was already dequeued and no new buffers are expected to become available.

ioctl VIDIOC_QUERYBUF

Name

VIDIOC_QUERYBUF - Query the status of a buffer

Synopsis

int **ioctl**(int *fd*, *VIDIOC_QUERYBUF*, struct *v4l2_buffer* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

This ioctl is part of the *streaming* I/O method. It can be used to query the status of a buffer at any time after buffers have been allocated with the *ioctl VIDIOC_REQBUFS* ioctl.

Applications set the `type` field of a struct *v4l2_buffer* to the same buffer type as was previously used with struct *v4l2_format* `type` and struct *v4l2_requestbuffers* `type`, and the `index` field. Valid index numbers range from zero to the number of buffers allocated with *ioctl VIDIOC_REQBUFS* (struct *v4l2_requestbuffers* `count`) minus one. The `reserved` and `reserved2` fields must be set to 0. When using the *multi-planar API*, the `m.planes` field must contain a userspace pointer to an array of struct *v4l2_plane* and the `length` field has to be set to the number of elements in that array. After calling *ioctl VIDIOC_QUERYBUF* with a pointer to this structure drivers return an error code or fill the rest of the structure.

In the `flags` field the `V4L2_BUF_FLAG_MAPPED`, `V4L2_BUF_FLAG_PREPARED`, `V4L2_BUF_FLAG_QUEUED` and `V4L2_BUF_FLAG_DONE` flags will be valid. The `memory` field will be set to the current I/O method. For the single-planar API, the `m.offset` contains the offset of the buffer from the start of the device memory, the `length` field its size. For the multi-planar API, fields `m.mem_offset` and `length` in the `m.planes` array elements will be used instead and the `length` field of struct *v4l2_buffer* is set to the number of filled-in array elements. The driver may or may not set the remaining fields and flags, they are meaningless in this context.

The struct *v4l2_buffer* structure is specified in *Buffers*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The buffer `type` is not supported, or the `index` is out of bounds.

ioctl VIDIOC_QUERYCAP

Name

VIDIOC_QUERYCAP - Query device capabilities

Synopsis

int **ioctl**(int *fd*, *VIDIOC_QUERYCAP*, struct *v4l2_capability* *argp)

Arguments

fd File descriptor returned by *open()*.

`argp`

Description

All V4L2 devices support the `VIDIOC_QUERYCAP` ioctl. It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities. The ioctl takes a pointer to a struct *v4l2_capability* which is filled by the driver. When the driver is not compatible with this specification the ioctl returns an `EINVAL` error code.

v4l2_capability

Table 1.140: struct v4l2_capability

__u8	driver[16]	Name of the driver, a unique NUL-terminated ASCII string. For example: “bttv”. Driver specific applications can use this information to verify the driver identity. It is also useful to work around known bugs, or to identify drivers in error reports. Storing strings in fixed sized arrays is bad practice but unavoidable here. Drivers and applications should take precautions to never read or write beyond the end of the array and to make sure the strings are properly NUL-terminated.
__u8	card[32]	Name of the device, a NUL-terminated UTF-8 string. For example: “Yoyodyne TV/FM”. One driver may support different brands or models of video hardware. This information is intended for users, for example in a menu of available devices. Since multiple TV cards of the same brand may be installed which are supported by the same driver, this name should be combined with the character device file name (e. g. /dev/video2) or the bus_info string to avoid ambiguities.
__u8	bus_info[32]	Location of the device in the system, a NUL-terminated ASCII string. For example: “PCI:0000:05:06.0”. This information is intended for users, to distinguish multiple identical devices. If no such information is available the field must simply count the devices controlled by the driver (“platform:vivi-000”). The bus_info must start with “PCI:” for PCI boards, “PCIe:” for PCI Express boards, “usb-” for USB devices, “I2C:” for i2c devices, “ISA:” for ISA devices, “parport” for parallel port devices and “platform:” for platform devices.
__u32	version	Version number of the driver. Starting with kernel 3.1, the version reported is provided by the V4L2 subsystem following the kernel numbering scheme. However, it may not always return the same version as the kernel if, for example, a stable or distribution-modified kernel uses the V4L2 stack from a newer kernel. The version number is formatted using the KERNEL_VERSION() macro:

`#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c))`
 	
`__u32 version = KERNEL_VERSION(0, 8, 1);`
 	
`printf ("Version: %u.%u.%u\\n",`
 	
`(version >> 16) & 0xFF, (version >> 8) & 0xFF, version & 0xFF);`		

__u32	capabilities	Available capabilities of the physical device as a whole, see *Device Capabilities Flags* . The same physical device can export multiple devices in /dev (e.g. /dev/videoX, /dev/vbiY and /dev/radioZ). The capabilities field should contain a union of all capabilities available around the several V4L2 devices exported to userspace. For all those devices the capabilities field returns the same set of capabilities. This allows applications to open just one of the devices (typically the video device) and discover whether video, vbi and/or radio are also supported.
__u32	device_caps	Device capabilities of the opened device, see *Device Capabilities Flags* . Should contain the available capabilities of that specific device node. So, for example, device_caps of a radio device will only contain radio related capabilities and no video or vbi capabilities. This field is only set if the capabilities field contains the V4L2_CAP_DEVICE_CAPS capability. Only the capabilities field can have the V4L2_CAP_DEVICE_CAPS capability, device_caps will never set V4L2_CAP_DEVICE_CAPS.
__u32	reserved[3]	Reserved for future extensions. Drivers must set this array to zero.

Table 1.141: Device Capabilities Flags

V4L2_CAP_VIDEO_CAPTURE	0x00000001	The device supports the single-planar API through the *Video Capture* interface.
V4L2_CAP_VIDEO_CAPTURE_MPLANE	0x00001000	The device supports the *multi-planar API* through the *Video Capture* interface.
V4L2_CAP_VIDEO_OUTPUT	0x00000002	The device supports the single-planar API through the *Video Output* interface.
V4L2_CAP_VIDEO_OUTPUT_MPLANE	0x00002000	The device supports the *multi-planar API* through the *Video Output* interface.
V4L2_CAP_VIDEO_M2M	0x00004000	The device supports the single-planar API through the Video Memory-To-Memory interface.
V4L2_CAP_VIDEO_M2M_MPLANE	0x00008000	The device supports the *multi-planar API* through the Video Memory-To-Memory interface.
V4L2_CAP_VIDEO_OVERLAY	0x00000004	The device supports the *Video Overlay* interface. A video overlay device typically stores captured images directly in the video memory of a graphics card, with hardware clipping and scaling.
V4L2_CAP_VBI_CAPTURE	0x00000010	The device supports the *Raw VBI Capture* interface, providing Teletext and Closed Caption data.
V4L2_CAP_VBI_OUTPUT	0x00000020	The device supports the *Raw VBI Output* interface.
V4L2_CAP_SLICED_VBI_CAPTURE	0x00000040	The device supports the *Sliced VBI Capture* interface.
V4L2_CAP_SLICED_VBI_OUTPUT	0x00000080	The device supports the *Sliced VBI Output* interface.
V4L2_CAP_RDS_CAPTURE	0x00000100	The device supports the *RDS* capture interface.
V4L2_CAP_VIDEO_OUTPUT_OVERLAY	0x00000200	The device supports the *Video Output Overlay* (OSD) interface. Unlike the *Video Overlay* interface, this is a secondary function of video output devices and overlays an image onto an outgoing video signal. When the driver sets this flag, it must clear the V4L2_CAP_VIDEO_OVERLAY flag and vice versa. [1]
V4L2_CAP_HW_FREQ_SEEK	0x00000400	The device supports the *ioctl VIDIOC_S_HW_FREQ_SEEK* ioctl for hardware frequency seeking.
V4L2_CAP_RDS_OUTPUT	0x00000800	The device supports the *RDS* output interface.
V4L2_CAP_TUNER	0x00010000	The device has some sort of tuner to receive RF-modulated video signals. For more information about tuner programming see *Tuners and Modulators* .
V4L2_CAP_AUDIO	0x00020000	The device has audio inputs or outputs. It may or may not support audio recording or playback, in PCM or compressed formats. PCM audio support must be implemented as ALSA or OSS interface. For more information on audio inputs and outputs see *Audio Inputs and Outputs* .
V4L2_CAP_RADIO	0x00040000	This is a radio receiver.
V4L2_CAP_MODULATOR	0x00080000	The device has some sort of modulator to emit RF-modulated video/audio signals. For more information about modulator programming see *Tuners and Modulators* .
V4L2_CAP_SDR_CAPTURE	0x00100000	The device supports the *SDR Capture* interface.
V4L2_CAP_EXT_PIX_FORMAT	0x00200000	The device supports the struct *v4l2_pix_format* extended fields.

Continued on next page

Table 1.141 – continued from previous page

V4L2_CAP_SDR_OUTPUT	0x00400000	The device supports the *SDR Output* interface.
V4L2_CAP_READWRITE	0x01000000	The device supports the *read()* and/or *write()* I/O methods.
V4L2_CAP_ASYNCIO	0x02000000	The device supports the *asynchronous* I/O methods.
V4L2_CAP_STREAMING	0x04000000	The device supports the *streaming* I/O method.
V4L2_CAP_TOUCH	0x10000000	This is a touch device.
V4L2_CAP_DEVICE_CAPS	0x80000000	The driver fills the device_caps field. This capability can only appear in the capabilities field and never in the device_caps field.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU

Name

VIDIOC_QUERYCTRL - VIDIOC_QUERY_EXT_CTRL - VIDIOC_QUERYMENU - Enumerate controls and menu control items

Synopsis

int **ioctl**(int *fd*, int *VIDIOC_QUERYCTRL*, struct *v4l2_queryctrl* *argp)

int **ioctl**(int *fd*, *VIDIOC_QUERY_EXT_CTRL*, struct *v4l2_query_ext_ctrl* *argp)

int **ioctl**(int *fd*, *VIDIOC_QUERYMENU*, struct *v4l2_querymenu* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the attributes of a control applications set the id field of a struct *v4l2_queryctrl* and call the VIDIOC_QUERYCTRL ioctl with a pointer to this structure. The driver fills the rest of the structure or returns an EINVAL error code when the id is invalid.

It is possible to enumerate controls by calling VIDIOC_QUERYCTRL with successive id values starting from V4L2_CID_BASE up to and exclusive V4L2_CID_LASTP1. Drivers may return EINVAL if a control in this range is not supported. Further applications can enumerate private controls, which are not defined in this specification, by starting at V4L2_CID_PRIVATE_BASE and incrementing id until the driver returns EINVAL.

[1] The struct *v4l2_framebuffer* lacks an enum *v4l2_buf_type* field, therefore the type of overlay is implied by the driver capabilities.

In both cases, when the driver sets the V4L2_CTRL_FLAG_DISABLED flag in the flags field this control is permanently disabled and should be ignored by the application. [1]

When the application ORs id with V4L2_CTRL_FLAG_NEXT_CTRL the driver returns the next supported non-compound control, or EINVAL if there is none. In addition, the V4L2_CTRL_FLAG_NEXT_COMPOUND flag can be specified to enumerate all compound controls (i.e. controls with type ≥ V4L2_CTRL_COMPOUND_TYPES and/or array control, in other words controls that contain more than one value). Specify both V4L2_CTRL_FLAG_NEXT_CTRL and V4L2_CTRL_FLAG_NEXT_COMPOUND in order to enumerate all controls, compound or not. Drivers which do not support these flags yet always return EINVAL.

The VIDIOC_QUERY_EXT_CTRL ioctl was introduced in order to better support controls that can use compound types, and to expose additional control information that cannot be returned in struct *v4l2_queryctrl* since that structure is full.

VIDIOC_QUERY_EXT_CTRL is used in the same way as VIDIOC_QUERYCTRL, except that the reserved array must be zeroed as well.

Additional information is required for menu controls: the names of the menu items. To query them applications set the id and index fields of struct *v4l2_querymenu* and call the VIDIOC_QUERYMENU ioctl with a pointer to this structure. The driver fills the rest of the structure or returns an EINVAL error code when the id or index is invalid. Menu items are enumerated by calling VIDIOC_QUERYMENU with successive index values from struct *v4l2_queryctrl* minimum to maximum, inclusive.

Note:

It is possible for VIDIOC_QUERYMENU to return an EINVAL error code for some indices between minimum and maximum. In that case that particular menu item is not supported by this driver. Also note that the minimum value is not necessarily 0.

See also the examples in *User Controls* .

Table 1.142: struct v4l2_queryctrl

__u32	id	Identifies the control, set by the application. See *Control IDs* for predefined IDs. When the ID is ORed with V4L2_CTRL_FLAG_NEXT_CTRL the driver clears the flag and returns the first control with a higher ID. Drivers which do not support this flag yet always return an EINVAL error code.
__u32	type	Type of control, see *v4l2_ctrl_type*.
__u8	name[32]	Name of the control, a NUL-terminated ASCII string. This information is intended for the user.
__s32	minimum	Minimum value, inclusive. This field gives a lower bound for the control. See enum *v4l2_ctrl_type* how the minimum value is to be used for each possible control type. Note that this a signed 32-bit value.
__s32	maximum	Maximum value, inclusive. This field gives an upper bound for the control. See enum *v4l2_ctrl_type* how the maximum value is to be used for each possible control type. Note that this a signed 32-bit value.

Continued on next page

[1] V4L2_CTRL_FLAG_DISABLED was intended for two purposes: Drivers can skip predefined controls not supported by the hardware (although returning EINVAL would do as well), or disable predefined and private controls after hardware detection without the trouble of reordering control arrays and indices (EINVAL cannot be used to skip private controls because it would prematurely end the enumeration).

Table 1.142 – continued from previous page

__s32	step	This field gives a step size for the control. See enum *v4l2_ctrl_type* how the step value is to be used for each possible control type. Note that this an unsigned 32-bit value. Generally drivers should not scale hardware control values. It may be necessary for example when the name or id imply a particular unit and the hardware actually accepts only multiples of said unit. If so, drivers must take care values are properly rounded when scaling, such that errors will not accumulate on repeated read-write cycles. This field gives the smallest change of an integer control actually affecting hardware. Often the information is needed when the user can change controls by keyboard or GUI buttons, rather than a slider. When for example a hardware register accepts values 0-511 and the driver reports 0-65535, step should be 128. Note that although signed, the step value is supposed to be always positive.
__s32	default_value	The default value of a V4L2_CTRL_TYPE_INTEGER, _BOOLEAN, _BITMASK, _MENU or _INTEGER_MENU control. Not valid for other types of controls. ***Note:*** *Drivers reset controls to their default value only when the driver is first loaded, never afterwards.*
__u32	flags	Control flags, see *Control Flags* .
__u32	reserved[2]	Reserved for future extensions. Drivers must set the array to zero.

Table 1.143: struct v4l2_query_ext_ctrl

__u32	id	Identifies the control, set by the application. See *Control IDs* for predefined IDs. When the ID is ORed with V4L2_CTRL_FLAG_NEXT_CTRL the driver clears the flag and returns the first non-compound control with a higher ID. When the ID is ORed with V4L2_CTRL_FLAG_NEXT_COMPOUND the driver clears the flag and returns the first compound control with a higher ID. Set both to get the first control (compound or not) with a higher ID.
__u32	type	Type of control, see *v4l2_ctrl_type*.
char	name[32]	Name of the control, a NUL-terminated ASCII string. This information is intended for the user.
__s64	minimum	Minimum value, inclusive. This field gives a lower bound for the control. See enum *v4l2_ctrl_type* how the minimum value is to be used for each possible control type. Note that this a signed 64-bit value.
__s64	maximum	Maximum value, inclusive. This field gives an upper bound for the control. See enum *v4l2_ctrl_type* how the maximum value is to be used for each possible control type. Note that this a signed 64-bit value.

Continued on next page

Table 1.143 – continued from previous page

__u64	step	This field gives a step size for the control. See enum *v4l2_ctrl_type* how the step value is to be used for each possible control type. Note that this an unsigned 64-bit value. Generally drivers should not scale hardware control values. It may be necessary for example when the name or id imply a particular unit and the hardware actually accepts only multiples of said unit. If so, drivers must take care values are properly rounded when scaling, such that errors will not accumulate on repeated read-write cycles. This field gives the smallest change of an integer control actually affecting hardware. Often the information is needed when the user can change controls by keyboard or GUI buttons, rather than a slider. When for example a hardware register accepts values 0-511 and the driver reports 0-65535, step should be 128.
__s64	default_value	The default value of a V4L2_CTRL_TYPE_INTEGER, _INTEGER64, _BOOLEAN, _BITMASK, _MENU, _INTEGER_MENU, _U8 or _U16 control. Not valid for other types of controls. ***Note:*** *Drivers reset controls to their default value only when the driver is first loaded, never afterwards.*
__u32	flags	Control flags, see *Control Flags* .
__u32	elem_size	The size in bytes of a single element of the array. Given a char pointer p to a 3-dimensional array you can find the position of cell (z,y,x) as follows: p + ((z * dims[1] + y) * dims[0] + x) * elem_size. elem_size is always valid, also when the control isn't an array. For string controls elem_size is equal to maximum + 1.
__u32	elems	The number of elements in the N-dimensional array. If this control is not an array, then elems is 1. The elems field can never be 0.
__u32	nr_of_dims	The number of dimension in the N-dimensional array. If this control is not an array, then this field is 0.
__u32	dims[V4L2_CTRL_MAX_DIMS]	The size of each dimension. The first nr_of_dims elements of this array must be non-zero, all remaining elements must be zero.
__u32	reserved[32]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 1.144: struct v4l2_querymenu

__u32		id	Identifies the control, set by the application from the respective struct *v4l2_queryctrl* id.
__u32		index	Index of the menu item, starting at zero, set by the application.
union			
	__u8	name[32]	Name of the menu item, a NUL-terminated ASCII string. This information is intended for the user. This field is valid for V4L2_CTRL_FLAG_MENU type controls.
	__s64	value	Value of the integer menu item. This field is valid for V4L2_CTRL_FLAG_INTEGER_MENU type controls.
__u32		reserved	Reserved for future extensions. Drivers must set the array to zero.

v4l2_ctrl_type

Table 1.145: enum v4l2_ctrl_type

Type	minimum	step	maximum	Description
V4L2_CTRL_TYPE_INTEGER	any	any	any	An integer-valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_BOOLEAN	0	1	1	A boolean-valued control. Zero corresponds to “disabled”, and one means “enabled”.
V4L2_CTRL_TYPE_MENU	≥ 0	1	N-1	The control has a menu of N choices. The names of the menu items can be enumerated with the VIDIOC_QUERYMENU ioctl.
V4L2_CTRL_TYPE_INTEGER_MENU	≥ 0	1	N-1	The control has a menu of N choices. The values of the menu items can be enumerated with the VIDIOC_QUERYMENU ioctl. This is similar to V4L2_CTRL_TYPE_MENU except that instead of strings, the menu items are signed 64-bit integers.
V4L2_CTRL_TYPE_BITMASK	0	n/a	any	A bitmask field. The maximum value is the set of bits that can be used, all other bits are to be 0. The maximum value is interpreted as a __u32, allowing the use of bit 31 in the bitmask.
V4L2_CTRL_TYPE_BUTTON	0	0	0	A control which performs an action when set. Drivers must ignore the value passed with VIDIOC_S_CTRL and return an EINVAL error code on a VIDIOC_G_CTRL attempt.
V4L2_CTRL_TYPE_INTEGER64	any	any	any	A 64-bit integer valued control. Minimum, maximum and step size cannot be queried using VIDIOC_QUERYCTRL. Only VIDIOC_QUERY_EXT_CTRL can retrieve the 64-bit min/max/step values, they should be interpreted as n/a when using VIDIOC_QUERYCTRL.
V4L2_CTRL_TYPE_STRING	≥ 0	≥ 1	≥ 0	The minimum and maximum string lengths. The step size means that the string must be (minimum + N * step) characters long for N ≥ 0. These lengths do not include the terminating zero, so in order to pass a string of length 8 to *VIDIOC_S_EXT_CTRLS* you need to set the size field of struct *v4l2_ext_control* to 9. For *VIDIOC_G_EXT_CTRLS* you can set the size field to maximum + 1. Which character encoding is used will depend on the string control itself and should be part of the control documentation.

Continued on next page

Table 1.145 – continued from previous page

Type	minimum	step	maximum	Description
V4L2_CTRL_TYPE_CTRL_CLASS	n/a	n/a	n/a	This is not a control. When VIDIOC_QUERYCTRL is called with a control ID equal to a control class code (see *Control classes*) + 1, the ioctl returns the name of the control class and this control type. Older drivers which do not support this feature return an EINVAL error code.
V4L2_CTRL_TYPE_U8	any	any	any	An unsigned 8-bit valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_U16	any	any	any	An unsigned 16-bit valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_U32	any	any	any	An unsigned 32-bit valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.

Table 1.146: Control Flags

V4L2_CTRL_FLAG_DISABLED	0x0001	This control is permanently disabled and should be ignored by the application. Any attempt to change the control will result in an EINVAL error code.
V4L2_CTRL_FLAG_GRABBED	0x0002	This control is temporarily unchangeable, for example because another application took over control of the respective resource. Such controls may be displayed specially in a user interface. Attempts to change the control may result in an EBUSY error code.
V4L2_CTRL_FLAG_READ_ONLY	0x0004	This control is permanently readable only. Any attempt to change the control will result in an EINVAL error code.
V4L2_CTRL_FLAG_UPDATE	0x0008	A hint that changing this control may affect the value of other controls within the same control class. Applications should update their user interface accordingly.
V4L2_CTRL_FLAG_INACTIVE	0x0010	This control is not applicable to the current configuration and should be displayed accordingly in a user interface. For example the flag may be set on a MPEG audio level 2 bitrate control when MPEG audio encoding level 1 was selected with another control.
V4L2_CTRL_FLAG_SLIDER	0x0020	A hint that this control is best represented as a slider-like element in a user interface.

Continued on next page

Table 1.146 – continued from previous page

V4L2_CTRL_FLAG_WRITE_ONLY	0x0040	This control is permanently writable only. Any attempt to read the control will result in an EACCES error code error code. This flag is typically present for relative controls or action controls where writing a value will cause the device to carry out a given action (e. g. motor control) but no meaningful value can be returned.
V4L2_CTRL_FLAG_VOLATILE	0x0080	This control is volatile, which means that the value of the control changes continuously. A typical example would be the current gain value if the device is in auto-gain mode. In such a case the hardware calculates the gain value based on the lighting conditions which can change over time. ***Note:*** *Setting a new value for a volatile control will be ignored unless* V4L2_CTRL_FLAG_EXECUTE_ON_WRITE *is also set. Setting a new value for a volatile control will* never *trigger a* V4L2_EVENT_CTRL_CH_VALUE *event.*
V4L2_CTRL_FLAG_HAS_PAYLOAD	0x0100	This control has a pointer type, so its value has to be accessed using one of the pointer fields of struct *v4l2_ext_control*. This flag is set for controls that are an array, string, or have a compound type. In all cases you have to set a pointer to memory containing the payload of the control.
V4L2_CTRL_FLAG_EXECUTE_ON_WRITE	0x0200	The value provided to the control will be propagated to the driver even if it remains constant. This is required when the control represents an action on the hardware. For example: clearing an error flag or triggering the flash. All the controls of the type V4L2_CTRL_TYPE_BUTTON have this flag set.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_queryctrl* `id` is invalid. The struct *v4l2_querymenu* `id` is invalid or `index` is out of range (less than `minimum` or greater than `maximum`) or this particular menu item is not supported by the driver.

EACCES An attempt was made to read a write-only control.

ioctl VIDIOC_QUERY_DV_TIMINGS

Name

VIDIOC_QUERY_DV_TIMINGS - VIDIOC_SUBDEV_QUERY_DV_TIMINGS - Sense the DV preset received by the current input

Synopsis

int **ioctl**(int *fd*, *VIDIOC_QUERY_DV_TIMINGS*, struct *v4l2_dv_timings* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_QUERY_DV_TIMINGS*, struct *v4l2_dv_timings* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

The hardware may be able to detect the current DV timings automatically, similar to sensing the video standard. To do so, applications call *ioctl VIDIOC_QUERY_DV_TIMINGS* with a pointer to a struct *v4l2_dv_timings*. Once the hardware detects the timings, it will fill in the timings structure.

Note:

Drivers shall not *switch timings automatically if new timings are detected. Instead, drivers should send the V4L2_EVENT_SOURCE_CHANGE event (if they support this) and expect that userspace will take action by calling* ioctl VIDIOC_QUERY_DV_TIMINGS *. The reason is that new timings usually mean different buffer sizes as well, and you cannot change buffer sizes on the fly. In general, applications that receive the Source Change event will have to call* ioctl VIDIOC_QUERY_DV_TIMINGS *, and if the detected timings are valid they will have to stop streaming, set the new timings, allocate new buffers and start streaming again.*

If the timings could not be detected because there was no signal, then ENOLINK is returned. If a signal was detected, but it was unstable and the receiver could not lock to the signal, then ENOLCK is returned. If the receiver could lock to the signal, but the format is unsupported (e.g. because the pixelclock is out of range of the hardware capabilities), then the driver fills in whatever timings it could find and returns ERANGE. In that case the application can call *ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP* to compare the found timings with the hardware's capabilities in order to give more precise feedback to the user.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA Digital video timings are not supported for this input or output.

ENOLINK No timings could be detected because no signal was found.

ENOLCK The signal was unstable and the hardware could not lock on to it.

ERANGE Timings were found, but they are out of range of the hardware capabilities.

ioctl VIDIOC_QUERYSTD

Name

VIDIOC_QUERYSTD - Sense the video standard received by the current input

Synopsis

int **ioctl**(int *fd*, *VIDIOC_QUERYSTD*, v4l2_std_id *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

The hardware may be able to detect the current video standard automatically. To do so, applications call *ioctl VIDIOC_QUERYSTD* with a pointer to a *v4l2_std_id* type. The driver stores here a set of candidates, this can be a single flag or a set of supported standards if for example the hardware can only distinguish between 50 and 60 Hz systems. If no signal was detected, then the driver will return V4L2_STD_UNKNOWN. When detection is not possible or fails, the set must contain all standards supported by the current video input or output.

Note:

Drivers shall not *switch the video standard automatically if a new video standard is detected. Instead, drivers should send the V4L2_EVENT_SOURCE_CHANGE event (if they support this) and expect that userspace will take action by calling* ioctl VIDIOC_QUERYSTD *. The reason is that a new video standard can mean different buffer sizes as well, and you cannot change buffer sizes on the fly. In general, applications that receive the Source Change event will have to call* ioctl VIDIOC_QUERYSTD *, and if the detected video standard is valid they will have to stop streaming, set the new standard, allocate new buffers and start streaming again.*

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA Standard video timings are not supported for this input or output.

ioctl VIDIOC_REQBUFS

Name

VIDIOC_REQBUFS - Initiate Memory Mapping, User Pointer I/O or DMA buffer I/O

Synopsis

int **ioctl**(int *fd*, *VIDIOC_REQBUFS*, struct *v4l2_requestbuffers* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

This ioctl is used to initiate *memory mapped* , *user pointer* or *DMABUF* based I/O. Memory mapped buffers are located in device memory and must be allocated with this ioctl before they can be mapped into the application's address space. User buffers are allocated by applications themselves, and this ioctl is merely used to switch the driver into user pointer I/O mode and to setup some internal structures. Similarly, DMABUF buffers are allocated by applications through a device driver, and this ioctl only configures the driver into DMABUF I/O mode without performing any direct allocation.

To allocate device buffers applications initialize all fields of the struct *v4l2_requestbuffers* structure. They set the `type` field to the respective stream or buffer type, the `count` field to the desired number of buffers, `memory` must be set to the requested I/O method and the `reserved` array must be zeroed. When the ioctl is called with a pointer to this structure the driver will attempt to allocate the requested number of buffers and it stores the actual number allocated in the `count` field. It can be smaller than the number requested, even zero, when the driver runs out of free memory. A larger number is also possible when the driver requires more buffers to function correctly. For example video output requires at least two buffers, one displayed and one filled by the application.

When the I/O method is not supported the ioctl returns an `EINVAL` error code.

Applications can call *ioctl VIDIOC_REQBUFS* again to change the number of buffers, however this cannot succeed when any buffers are still mapped. A `count` value of zero frees all buffers, after aborting or finishing any DMA in progress, an implicit *VIDIOC_STREAMOFF* .

`v4l2_requestbuffers`

Table 1.147: struct v4l2_requestbuffers

__u32	`count`	The number of buffers requested or granted.
__u32	`type`	Type of the stream or buffers, this is the same as the struct *v4l2_format* `type` field. See *v4l2_buf_type* for valid values.
__u32	`memory`	Applications set this field to `V4L2_MEMORY_MMAP`, `V4L2_MEMORY_DMABUF` or `V4L2_MEMORY_USERPTR`. See *v4l2_memory*.
__u32	`reserved[2]`	A place holder for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The buffer type (`type` field) or the requested I/O method (`memory`) is not supported.

ioctl VIDIOC_S_HW_FREQ_SEEK

Name

VIDIOC_S_HW_FREQ_SEEK - Perform a hardware frequency seek

Synopsis

int **`ioctl`**(int *fd*, *VIDIOC_S_HW_FREQ_SEEK*, struct *v4l2_hw_freq_seek* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Start a hardware frequency seek from the current frequency. To do this applications initialize the tuner, type, seek_upward, wrap_around, spacing, rangelow and rangehigh fields, and zero out the reserved array of a struct *v4l2_hw_freq_seek* and call the VIDIOC_S_HW_FREQ_SEEK ioctl with a pointer to this structure.

The rangelow and rangehigh fields can be set to a non-zero value to tell the driver to search a specific band. If the struct *v4l2_tuner* capability field has the V4L2_TUNER_CAP_HWSEEK_PROG_LIM flag set, these values must fall within one of the bands returned by *ioctl VIDIOC_ENUM_FREQ_BANDS* . If the V4L2_TUNER_CAP_HWSEEK_PROG_LIM flag is not set, then these values must exactly match those of one of the bands returned by *ioctl VIDIOC_ENUM_FREQ_BANDS* . If the current frequency of the tuner does not fall within the selected band it will be clamped to fit in the band before the seek is started.

If an error is returned, then the original frequency will be restored.

This ioctl is supported if the V4L2_CAP_HW_FREQ_SEEK capability is set.

If this ioctl is called from a non-blocking filehandle, then EAGAIN error code is returned and no seek takes place.

v4l2_hw_freq_seek

Table 1.148: struct v4l2_hw_freq_seek

__u32	tuner	The tuner index number. This is the same value as in the struct *v4l2_input* tuner field and the struct *v4l2_tuner* index field.
__u32	type	The tuner type. This is the same value as in the struct *v4l2_tuner* type field. See *v4l2_tuner_type*
__u32	seek_upward	If non-zero, seek upward from the current frequency, else seek downward.
__u32	wrap_around	If non-zero, wrap around when at the end of the frequency range, else stop seeking. The struct *v4l2_tuner* capability field will tell you what the hardware supports.
__u32	spacing	If non-zero, defines the hardware seek resolution in Hz. The driver selects the nearest value that is supported by the device. If spacing is zero a reasonable default value is used.
__u32	rangelow	If non-zero, the lowest tunable frequency of the band to search in units of 62.5 kHz, or if the struct *v4l2_tuner* capability field has the V4L2_TUNER_CAP_LOW flag set, in units of 62.5 Hz or if the struct *v4l2_tuner* capability field has the V4L2_TUNER_CAP_1HZ flag set, in units of 1 Hz. If rangelow is zero a reasonable default value is used.
__u32	rangehigh	If non-zero, the highest tunable frequency of the band to search in units of 62.5 kHz, or if the struct *v4l2_tuner* capability field has the V4L2_TUNER_CAP_LOW flag set, in units of 62.5 Hz or if the struct *v4l2_tuner* capability field has the V4L2_TUNER_CAP_1HZ flag set, in units of 1 Hz. If rangehigh is zero a reasonable default value is used.
__u32	reserved[5]	Reserved for future extensions. Applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The tuner index is out of bounds, the wrap_around value is not supported or one of the values in the type, rangelow or rangehigh fields is wrong.

EAGAIN Attempted to call VIDIOC_S_HW_FREQ_SEEK with the filehandle in non-blocking mode.

ENODATA The hardware seek found no channels.

EBUSY Another hardware seek is already in progress.

ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF

Name

VIDIOC_STREAMON - VIDIOC_STREAMOFF - Start or stop streaming I/O

Synopsis

int **ioctl**(int *fd*, *VIDIOC_STREAMON*, const int *argp)

int **ioctl**(int *fd*, *VIDIOC_STREAMOFF*, const int *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

The VIDIOC_STREAMON and VIDIOC_STREAMOFF ioctl start and stop the capture or output process during streaming (*memory mapping* , *user pointer* or *DMABUF*) I/O.

Capture hardware is disabled and no input buffers are filled (if there are any empty buffers in the incoming queue) until VIDIOC_STREAMON has been called. Output hardware is disabled and no video signal is produced until VIDIOC_STREAMON has been called. The ioctl will succeed when at least one output buffer is in the incoming queue.

Memory-to-memory devices will not start until VIDIOC_STREAMON has been called for both the capture and output stream types.

If VIDIOC_STREAMON fails then any already queued buffers will remain queued.

The VIDIOC_STREAMOFF ioctl, apart of aborting or finishing any DMA in progress, unlocks any user pointer buffers locked in physical memory, and it removes all buffers from the incoming and outgoing queues. That means all images captured but not dequeued yet will be lost, likewise all images enqueued for output but not transmitted yet. I/O returns to the same state as after calling *ioctl VIDIOC_REQBUFS* and can be restarted accordingly.

If buffers have been queued with *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* and VIDIOC_STREAMOFF is called without ever having called VIDIOC_STREAMON, then those queued buffers will also be removed from the incoming queue and all are returned to the same state as after calling *ioctl VIDIOC_REQBUFS* and can be restarted accordingly.

Both ioctls take a pointer to an integer, the desired buffer or stream type. This is the same as struct *v4l2_requestbuffers* type.

If VIDIOC_STREAMON is called when streaming is already in progress, or if VIDIOC_STREAMOFF is called when streaming is already stopped, then 0 is returned. Nothing happens in the case of VIDIOC_STREAMON, but VIDIOC_STREAMOFF will return queued buffers to their starting state as mentioned above.

Note:

Applications can be preempted for unknown periods right before or after the VIDIOC_STREAMON or VIDIOC_STREAMOFF calls, there is no notion of starting or stopping "now". Buffer timestamps can be used to synchronize with other events.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The buffer type is not supported, or no buffers have been allocated (memory mapping) or enqueued (output) yet.

EPIPE The driver implements *pad-level format configuration* and the pipeline configuration is invalid.

ENOLINK The driver implements Media Controller interface and the pipeline link configuration is invalid.

ioctl VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL

Name

VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL - Enumerate frame intervals

Synopsis

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL*, struct *v4l2_subdev_frame_interval_enum* * *argp*)

Arguments

fd File descriptor returned by *open()* .

argp

Description

This ioctl lets applications enumerate available frame intervals on a given sub-device pad. Frame intervals only makes sense for sub-devices that can control the frame period on their own. This includes, for instance, image sensors and TV tuners.

For the common use case of image sensors, the frame intervals available on the sub-device output pad depend on the frame format and size on the same pad. Applications must thus specify the desired format and size when enumerating frame intervals.

To enumerate frame intervals applications initialize the index, pad, which, code, width and height fields of struct *v4l2_subdev_frame_interval_enum* and call the *ioctl VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code if one of the input fields is invalid. All frame intervals are enumerable by beginning at index zero and incrementing by one until EINVAL is returned.

Available frame intervals may depend on the current 'try' formats at other pads of the sub-device, as well as on the current active links. See *ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT* for more information about the try formats.

Sub-devices that support the frame interval enumeration ioctl should implemented it on a single pad only. Its behaviour when supported on multiple pads of the same sub-device is not defined.

v4l2_subdev_frame_interval_enum

Table 1.149: struct v4l2_subdev_frame_interval_enum

__u32	index	Number of the format in the enumeration, set by the application.
__u32	pad	Pad number as reported by the media controller API.
__u32	code	The media bus format code, as defined in *Media Bus Formats* .
__u32	width	Frame width, in pixels.
__u32	height	Frame height, in pixels.
struct *v4l2_fract*	interval	Period, in seconds, between consecutive video frames.
__u32	which	Frame intervals to be enumerated, from enum *v4l2_subdev_format_whence* .
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_subdev_frame_interval_enum* pad references a non-existing pad, one of the code, `width` or `height` fields are invalid for the given pad or the `index` field is out of bounds.

ioctl VIDIOC_SUBDEV_ENUM_FRAME_SIZE

Name

VIDIOC_SUBDEV_ENUM_FRAME_SIZE - Enumerate media bus frame sizes

Synopsis

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_ENUM_FRAME_SIZE*, struct *v4l2_subdev_frame_size_enum* * *argp*)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

This ioctl allows applications to enumerate all frame sizes supported by a sub-device on the given pad for the given media bus format. Supported formats can be retrieved with the *ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE* ioctl.

To enumerate frame sizes applications initialize the pad, `which` , `code` and `index` fields of the struct *v4l2_subdev_mbus_code_enum* and call the *ioctl VIDIOC_SUBDEV_ENUM_FRAME_SIZE* ioctl with a pointer to the structure. Drivers fill the minimum and maximum frame sizes or return an EINVAL error code if one of the input parameters is invalid.

Sub-devices that only support discrete frame sizes (such as most sensors) will return one or more frame sizes with identical minimum and maximum values.

Not all possible sizes in given [minimum, maximum] ranges need to be supported. For instance, a scaler that uses a fixed-point scaling ratio might not be able to produce every frame size between the minimum and maximum values. Applications must use the *VIDIOC_SUBDEV_S_FMT* ioctl to try the sub-device for an exact supported frame size.

Available frame sizes may depend on the current 'try' formats at other pads of the sub-device, as well as on the current active links and the current values of V4L2 controls. See *ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT* for more information about try formats.

v4l2_subdev_frame_size_enum

Table 1.150: struct v4l2_subdev_frame_size_enum

__u32	index	Number of the format in the enumeration, set by the application.
__u32	pad	Pad number as reported by the media controller API.
__u32	code	The media bus format code, as defined in *Media Bus Formats* .
__u32	min_width	Minimum frame width, in pixels.
__u32	max_width	Maximum frame width, in pixels.
__u32	min_height	Minimum frame height, in pixels.
__u32	max_height	Maximum frame height, in pixels.
__u32	which	Frame sizes to be enumerated, from enum *v4l2_subdev_format_whence* .
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_subdev_frame_size_enum* pad references a non-existing pad, the `code` is invalid for the given pad or the `index` field is out of bounds.

ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE

Name

VIDIOC_SUBDEV_ENUM_MBUS_CODE - Enumerate media bus formats

Synopsis

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_ENUM_MBUS_CODE*, struct *v4l2_subdev_mbus_code_enum* * *argp*)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

To enumerate media bus formats available at a given sub-device pad applications initialize the pad, which and index fields of struct *v4l2_subdev_mbus_code_enum* and call the *ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code if either the pad or index are invalid. All media bus formats are enumerable by beginning at index zero and incrementing by one until EINVAL is returned.

Available media bus formats may depend on the current 'try' formats at other pads of the sub-device, as well as on the current active links. See *ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT* for more information about the try formats.

v4l2_subdev_mbus_code_enum

Table 1.151: struct v4l2_subdev_mbus_code_enum

__u32	pad	Pad number as reported by the media controller API.
__u32	index	Number of the format in the enumeration, set by the application.
__u32	code	The media bus format code, as defined in *Media Bus Formats* .
__u32	which	Media bus format codes to be enumerated, from enum *v4l2_subdev_format_whence* .
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *v4l2_subdev_mbus_code_enum* pad references a non-existing pad, or the index field is out of bounds.

ioctl VIDIOC_SUBDEV_G_CROP, VIDIOC_SUBDEV_S_CROP

Name

VIDIOC_SUBDEV_G_CROP - VIDIOC_SUBDEV_S_CROP - Get or set the crop rectangle on a subdev pad

Synopsis

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_G_CROP*, struct *v4l2_subdev_crop* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_S_CROP*, const struct *v4l2_subdev_crop* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Note:

This is an Obsolete API Elements *interface and may be removed in the future. It is superseded by* the selection API .

To retrieve the current crop rectangle applications set the pad field of a struct *v4l2_subdev_crop* to the desired pad number as reported by the media API and the `which` field to `V4L2_SUBDEV_FORMAT_ACTIVE`. They then call the `VIDIOC_SUBDEV_G_CROP` ioctl with a pointer to this structure. The driver fills the members of the `rect` field or returns `EINVAL` error code if the input arguments are invalid, or if cropping is not supported on the given pad.

To change the current crop rectangle applications set both the pad and `which` fields and all members of the `rect` field. They then call the `VIDIOC_SUBDEV_S_CROP` ioctl with a pointer to this structure. The driver verifies the requested crop rectangle, adjusts it based on the hardware capabilities and configures the device. Upon return the struct *v4l2_subdev_crop* contains the current format as would be returned by a `VIDIOC_SUBDEV_G_CROP` call.

Applications can query the device capabilities by setting the `which` to `V4L2_SUBDEV_FORMAT_TRY`. When set, 'try' crop rectangles are not applied to the device by the driver, but are mangled exactly as active crop rectangles and stored in the sub-device file handle. Two applications querying the same sub-device would thus not interact with each other.

Drivers must not return an error solely because the requested crop rectangle doesn't match the device capabilities. They must instead modify the rectangle to match what the hardware can provide. The modified format should be as close as possible to the original request.

v4l2_subdev_crop

Table 1.152: struct v4l2_subdev_crop

__u32	pad	Pad number as reported by the media framework.
__u32	which	Crop rectangle to get or set, from enum *v4l2_subdev_format_whence* .
struct *v4l2_rect*	rect	Crop rectangle boundaries, in pixels.
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY The crop rectangle can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The ioctl must not be retried without performing another action to fix the problem first. Only returned by `VIDIOC_SUBDEV_S_CROP`

EINVAL The struct *v4l2_subdev_crop* pad references a non-existing pad, the `which` field references a non-existing format, or cropping is not supported on the given subdev pad.

ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT

Name

VIDIOC_SUBDEV_G_FMT - VIDIOC_SUBDEV_S_FMT - Get or set the data format on a subdev pad

Synopsis

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_G_FMT*, struct *v4l2_subdev_format* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_S_FMT*, struct *v4l2_subdev_format* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

These ioctls are used to negotiate the frame format at specific subdev pads in the image pipeline.

To retrieve the current format applications set the pad field of a struct *v4l2_subdev_format* to the desired pad number as reported by the media API and the which field to V4L2_SUBDEV_FORMAT_ACTIVE. When they call the VIDIOC_SUBDEV_G_FMT ioctl with a pointer to this structure the driver fills the members of the format field.

To change the current format applications set both the pad and which fields and all members of the format field. When they call the VIDIOC_SUBDEV_S_FMT ioctl with a pointer to this structure the driver verifies the requested format, adjusts it based on the hardware capabilities and configures the device. Upon return the struct *v4l2_subdev_format* contains the current format as would be returned by a VIDIOC_SUBDEV_G_FMT call.

Applications can query the device capabilities by setting the which to V4L2_SUBDEV_FORMAT_TRY. When set, 'try' formats are not applied to the device by the driver, but are changed exactly as active formats and stored in the sub-device file handle. Two applications querying the same sub-device would thus not interact with each other.

For instance, to try a format at the output pad of a sub-device, applications would first set the try format at the sub-device input with the VIDIOC_SUBDEV_S_FMT ioctl. They would then either retrieve the default format at the output pad with the VIDIOC_SUBDEV_G_FMT ioctl, or set the desired output pad format with the VIDIOC_SUBDEV_S_FMT ioctl and check the returned value.

Try formats do not depend on active formats, but can depend on the current links configuration or sub-device controls value. For instance, a low-pass noise filter might crop pixels at the frame boundaries, modifying its output frame size.

Drivers must not return an error solely because the requested format doesn't match the device capabilities. They must instead modify the format to match what the hardware can provide. The modified format should be as close as possible to the original request.

v4l2_subdev_format

Table 1.153: struct v4l2_subdev_format

__u32	pad	Pad number as reported by the media controller API.
__u32	which	Format to modified, from enum *v4l2_subdev_format_whence* .
struct *v4l2_mbus_framefmt*	format	Definition of an image format, see *v4l2_mbus_framefmt* for details.
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 1.154: enum v4l2_subdev_format_whence

V4L2_SUBDEV_FORMAT_TRY	0	Try formats, used for querying device capabilities.
V4L2_SUBDEV_FORMAT_ACTIVE	1	Active formats, applied to the hardware.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY The format can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The ioctl must not be retried without performing another action to fix the problem first. Only returned by `VIDIOC_SUBDEV_S_FMT`

EINVAL The struct *v4l2_subdev_format* pad references a non-existing pad, or the `which` field references a non-existing format.

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_SUBDEV_G_FRAME_INTERVAL, VIDIOC_SUBDEV_S_FRAME_INTERVAL

Name

VIDIOC_SUBDEV_G_FRAME_INTERVAL - VIDIOC_SUBDEV_S_FRAME_INTERVAL - Get or set the frame interval on a subdev pad

Synopsis

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_G_FRAME_INTERVAL*, struct *v4l2_subdev_frame_interval* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_S_FRAME_INTERVAL*, struct *v4l2_subdev_frame_interval* *argp)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

These ioctls are used to get and set the frame interval at specific subdev pads in the image pipeline. The frame interval only makes sense for sub-devices that can control the frame period on their own. This includes, for instance, image sensors and TV tuners. Sub-devices that don't support frame intervals must not implement these ioctls.

To retrieve the current frame interval applications set the pad field of a struct *v4l2_subdev_frame_interval* to the desired pad number as reported by the media controller API. When they call the `VIDIOC_SUBDEV_G_FRAME_INTERVAL` ioctl with a pointer to this structure the driver fills the members of the `interval` field.

To change the current frame interval applications set both the pad field and all members of the `interval` field. When they call the `VIDIOC_SUBDEV_S_FRAME_INTERVAL` ioctl with a pointer to this structure the driver verifies the requested interval, adjusts it based on the hardware capabilities and configures the device.

Upon return the struct *v4l2_subdev_frame_interval* contains the current frame interval as would be returned by a VIDIOC_SUBDEV_G_FRAME_INTERVAL call.

Drivers must not return an error solely because the requested interval doesn't match the device capabilities. They must instead modify the interval to match what the hardware can provide. The modified interval should be as close as possible to the original request.

Sub-devices that support the frame interval ioctls should implement them on a single pad only. Their behaviour when supported on multiple pads of the same sub-device is not defined.

v4l2_subdev_frame_interval

Table 1.155: struct v4l2_subdev_frame_interval

__u32	pad	Pad number as reported by the media controller API.
struct *v4l2_fract*	interval	Period, in seconds, between consecutive video frames.
__u32	reserved[9]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY The frame interval can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The ioctl must not be retried without performing another action to fix the problem first. Only returned by VIDIOC_SUBDEV_S_FRAME_INTERVAL

EINVAL The struct *v4l2_subdev_frame_interval* pad references a non-existing pad, or the pad doesn't support frame intervals.

ioctl VIDIOC_SUBDEV_G_SELECTION, VIDIOC_SUBDEV_S_SELECTION

Name

VIDIOC_SUBDEV_G_SELECTION - VIDIOC_SUBDEV_S_SELECTION - Get or set selection rectangles on a subdev pad

Synopsis

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_G_SELECTION*, struct *v4l2_subdev_selection* *argp)

int **ioctl**(int *fd*, *VIDIOC_SUBDEV_S_SELECTION*, struct *v4l2_subdev_selection* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

The selections are used to configure various image processing functionality performed by the subdevs which affect the image size. This currently includes cropping, scaling and composition.

The selection API replaces *the old subdev crop API* . All the function of the crop API, and more, are supported by the selections API.

See *Sub-device Interface* for more information on how each selection target affects the image processing pipeline inside the subdevice.

Types of selection targets

There are two types of selection targets: actual and bounds. The actual targets are the targets which configure the hardware. The BOUNDS target will return a rectangle that contain all possible actual rectangles.

Discovering supported features

To discover which targets are supported, the user can perform `VIDIOC_SUBDEV_G_SELECTION` on them. Any unsupported target will return `EINVAL`.

Selection targets and flags are documented in *Common selection definitions* .

`v4l2_subdev_selection`

Table 1.156: struct v4l2_subdev_selection

__u32	`which`	Active or try selection, from enum *v4l2_subdev_format_whence* .
__u32	`pad`	Pad number as reported by the media framework.
__u32	`target`	Target selection rectangle. See *Common selection definitions* .
__u32	`flags`	Flags. See *Selection flags* .
struct *v4l2_rect*	`r`	Selection rectangle, in pixels.
__u32	`reserved[8]`	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY The selection rectangle can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The ioctl must not be retried without performing another action to fix the problem first. Only returned by `VIDIOC_SUBDEV_S_SELECTION`

EINVAL The struct *v4l2_subdev_selection* pad references a non-existing pad, the `which` field references a non-existing format, or the selection target is not supported on the given subdev pad.

ioctl VIDIOC_SUBSCRIBE_EVENT, VIDIOC_UNSUBSCRIBE_EVENT

Name

VIDIOC_SUBSCRIBE_EVENT - VIDIOC_UNSUBSCRIBE_EVENT - Subscribe or unsubscribe event

Synopsis

int **`ioctl`**(int *fd*, *VIDIOC_SUBSCRIBE_EVENT*, struct *v4l2_event_subscription* *argp)

int **`ioctl`**(int *fd*, *VIDIOC_UNSUBSCRIBE_EVENT*, struct *v4l2_event_subscription* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Subscribe or unsubscribe V4L2 event. Subscribed events are dequeued by using the *ioctl VIDIOC_DQEVENT* ioctl.

v4l2_event_subscription

Table 1.157: struct v4l2_event_subscription

__u32	type	Type of the event, see *Event Types* . ***Note:*** *V4L2_EVENT_ALL can be used with* VIDIOC_UNSUBSCRIBE_EVENT *for unsubscribing all events at once.*
__u32	id	ID of the event source. If there is no ID associated with the event source, then set this to 0. Whether or not an event needs an ID depends on the event type.
__u32	flags	Event flags, see *Event Flags* .
__u32	reserved[5]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 1.158: Event Flags

V4L2_EVENT_SUB_FL_SEND_INITIAL	0x0001	When this event is subscribed an initial event will be sent containing the current status. This only makes sense for events that are triggered by a status change such as V4L2_EVENT_CTRL. Other events will ignore this flag.
V4L2_EVENT_SUB_FL_ALLOW_FEEDBACK	0x0002	If set, then events directly caused by an ioctl will also be sent to the filehandle that called that ioctl. For example, changing a control using *VIDIOC_S_CTRL* will cause a V4L2_EVENT_CTRL to be sent back to that same filehandle. Normally such events are suppressed to prevent feedback loops where an application changes a control to a one value and then another, and then receives an event telling it that that control has changed to the first value. Since it can't tell whether that event was caused by another application or by the *VIDIOC_S_CTRL* call it is hard to decide whether to set the control to the value in the event, or ignore it. Think carefully when you set this flag so you won't get into situations like that.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

V4L2 mmap()

Name

v4l2-mmap - Map device memory into application address space

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

void *__mmap__(void *_start_, size_t _length_, int _prot_, int _flags_, int _fd_, off_t _offset_)

Arguments

`start` Map the buffer to this address in the application's address space. When the `MAP_FIXED` flag is specified, `start` must be a multiple of the pagesize and mmap will fail when the specified address cannot be used. Use of this option is discouraged; applications should just specify a `NULL` pointer here.

`length` Length of the memory area to map. This must be the same value as returned by the driver in the struct *`v4l2_buffer`* `length` field for the single-planar API, and the same value as returned by the driver in the struct *`v4l2_plane`* `length` field for the multi-planar API.

`prot` The `prot` argument describes the desired memory protection. Regardless of the device type and the direction of data exchange it should be set to `PROT_READ | PROT_WRITE`, permitting read and write access to image buffers. Drivers should support at least this combination of flags.

Note:

1. *The Linux `videobuf` kernel module, which is used by some drivers supports only `PROT_READ | PROT_WRITE`. When the driver does not support the desired protection, the* mmap() *function fails.*
2. *Device memory accesses (e. g. the memory on a graphics card with video capturing hardware) may incur a performance penalty compared to main memory accesses, or reads may be significantly slower than writes or vice versa. Other I/O methods may be more efficient in such case.*

`flags` The `flags` parameter specifies the type of the mapped object, mapping options and whether modifications made to the mapped copy of the page are private to the process or are to be shared with other references.

`MAP_FIXED` requests that the driver selects no other address than the one specified. If the specified address cannot be used, *mmap()* will fail. If `MAP_FIXED` is specified, `start` must be a multiple of the pagesize. Use of this option is discouraged.

One of the `MAP_SHARED` or `MAP_PRIVATE` flags must be set. `MAP_SHARED` allows applications to share the mapped memory with other (e. g. child-) processes.

Note:

The Linux videobuf module which is used by some drivers supports only MAP_SHARED. MAP_PRIVATE requests copy-on-write semantics. V4L2 applications should not set the MAP_PRIVATE, MAP_DENYWRITE, MAP_EXECUTABLE or MAP_ANON flags.

fd File descriptor returned by *open()* .

offset Offset of the buffer in device memory. This must be the same value as returned by the driver in the struct *v4l2_buffer* `m` union `offset` field for the single-planar API, and the same value as returned by the driver in the struct *v4l2_plane* `m` union `mem_offset` field for the multi-planar API.

Description

The *mmap()* function asks to map `length` bytes starting at `offset` in the memory of the device specified by `fd` into the application address space, preferably at address `start`. This latter address is a hint only, and is usually specified as 0.

Suitable length and offset parameters are queried with the *ioctl VIDIOC_QUERYBUF* ioctl. Buffers must be allocated with the *ioctl VIDIOC_REQBUFS* ioctl before they can be queried.

To unmap buffers the *munmap()* function is used.

Return Value

On success *mmap()* returns a pointer to the mapped buffer. On error `MAP_FAILED` (-1) is returned, and the `errno` variable is set appropriately. Possible error codes are:

EBADF `fd` is not a valid file descriptor.

EACCES `fd` is not open for reading and writing.

EINVAL The `start` or `length` or `offset` are not suitable. (E. g. they are too large, or not aligned on a `PAGESIZE` boundary.)

The `flags` or `prot` value is not supported.

No buffers have been allocated with the *ioctl VIDIOC_REQBUFS* ioctl.

ENOMEM Not enough physical or virtual memory was available to complete the request.

V4L2 munmap()

Name

v4l2-munmap - Unmap device memory

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

int **munmap**(void **start*, size_t *length*)

Arguments

start Address of the mapped buffer as returned by the *mmap()* function.

length Length of the mapped buffer. This must be the same value as given to *mmap()* and returned by the driver in the struct *v4l2_buffer* `length` field for the single-planar API and in the struct *v4l2_plane* `length` field for the multi-planar API.

Description

Unmaps a previously with the *mmap()* function mapped buffer and frees it, if possible.

Return Value

On success *munmap()* returns 0, on failure -1 and the `errno` variable is set appropriately:

EINVAL The `start` or `length` is incorrect, or no buffers have been mapped yet.

V4L2 open()

Name

v4l2-open - Open a V4L2 device

Synopsis

```
#include <fcntl.h>
```

int **open**(const char ***device_name*, int *flags*)

Arguments

device_name Device to be opened.

flags Open flags. Access mode must be `O_RDWR`. This is just a technicality, input devices still support only reading and output devices only writing.

When the `O_NONBLOCK` flag is given, the *read()* function and the *VIDIOC_DQBUF* ioctl will return the `EAGAIN` error code when no data is available or no buffer is in the driver outgoing queue, otherwise these functions block until data becomes available. All V4L2 drivers exchanging data with applications must support the `O_NONBLOCK` flag.

Other flags have no effect.

Description

To open a V4L2 device applications call *open()* with the desired device name. This function has no side effects; all data format parameters, current input or output, control values or other properties remain unchanged. At the first *open()* call after loading the driver they will be reset to default values, drivers are never in an undefined state.

Return Value

On success *open()* returns the new file descriptor. On error -1 is returned, and the `errno` variable is set appropriately. Possible error codes are:

EACCES The caller has no permission to access the device.

EBUSY The driver does not support multiple opens and the device is already in use.

ENXIO No device corresponding to this device special file exists.

ENOMEM Not enough kernel memory was available to complete the request.

EMFILE The process already has the maximum number of files open.

ENFILE The limit on the total number of files open on the system has been reached.

V4L2 poll()

Name

v4l2-poll - Wait for some event on a file descriptor

Synopsis

```
#include <sys/poll.h>
```

int **poll**(struct pollfd **ufds*, unsigned int *nfds*, int *timeout*)

Arguments

Description

With the *poll()* function applications can suspend execution until the driver has captured data or is ready to accept data for output.

When streaming I/O has been negotiated this function waits until a buffer has been filled by the capture device and can be dequeued with the *VIDIOC_DQBUF* ioctl. For output devices this function waits until the device is ready to accept a new buffer to be queued up with the *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* ioctl for display. When buffers are already in the outgoing queue of the driver (capture) or the incoming queue isn't full (display) the function returns immediately.

On success *poll()* returns the number of file descriptors that have been selected (that is, file descriptors for which the `revents` field of the respective `struct pollfd()` structure is non-zero). Capture devices set the `POLLIN` and `POLLRDNORM` flags in the `revents` field, output devices the `POLLOUT` and `POLLWRNORM` flags. When the function timed out it returns a value of zero, on failure it returns -1 and the `errno` variable is set appropriately. When the application did not call *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* the *poll()* function succeeds, but sets the `POLLERR` flag in the `revents` field. When the application has called *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* for a capture device but hasn't yet called *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* , the *poll()* function succeeds and sets the `POLLERR` flag in the `revents` field. For output devices this same situation will cause *poll()* to succeed as well, but it sets the `POLLOUT` and `POLLWRNORM` flags in the `revents` field.

If an event occurred (see *ioctl VIDIOC_DQEVENT*) then `POLLPRI` will be set in the `revents` field and *poll()* will return.

When use of the *read()* function has been negotiated and the driver does not capture yet, the *poll()* function starts capturing. When that fails it returns a `POLLERR` as above. Otherwise it waits until data has

been captured and can be read. When the driver captures continuously (as opposed to, for example, still images) the function may return immediately.

When use of the *write()* function has been negotiated and the driver does not stream yet, the *poll()* function starts streaming. When that fails it returns a POLLERR as above. Otherwise it waits until the driver is ready for a non-blocking *write()* call.

If the caller is only interested in events (just POLLPRI is set in the events field), then *poll()* will *not* start streaming if the driver does not stream yet. This makes it possible to just poll for events and not for buffers.

All drivers implementing the *read()* or *write()* function or streaming I/O must also support the *poll()* function.

For more details see the *poll()* manual page.

Return Value

On success, *poll()* returns the number structures which have non-zero revents fields, or zero if the call timed out. On error -1 is returned, and the errno variable is set appropriately:

EBADF One or more of the ufds members specify an invalid file descriptor.

EBUSY The driver does not support multiple read or write streams and the device is already in use.

EFAULT ufds references an inaccessible memory area.

EINTR The call was interrupted by a signal.

EINVAL The nfds argument is greater than OPEN_MAX.

V4L2 read()

Name

v4l2-read - Read from a V4L2 device

Synopsis

```
#include <unistd.h>
```

ssize_t **read**(int *fd*, void **buf*, size_t *count*)

Arguments

fd File descriptor returned by *open()* .

buf Buffer to be filled

count Max number of bytes to read

Description

read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf. The layout of the data in the buffer is discussed in the respective device interface section, see ##. If count is zero, *read()* returns zero and has no other results. If count is greater than SSIZE_MAX, the result is unspecified. Regardless of the count value each *read()* call will provide at most one frame (two fields) worth of data.

By default *read()* blocks until data becomes available. When the O_NONBLOCK flag was given to the *open()* function it returns immediately with an EAGAIN error code when no data is available. The *select()* or *poll()* functions can always be used to suspend execution until data becomes available. All drivers supporting the *read()* function must also support *select()* and *poll()* .

Drivers can implement read functionality in different ways, using a single or multiple buffers and discarding the oldest or newest frames once the internal buffers are filled.

read() never returns a "snapshot" of a buffer being filled. Using a single buffer the driver will stop capturing when the application starts reading the buffer until the read is finished. Thus only the period of the vertical blanking interval is available for reading, or the capture rate must fall below the nominal frame rate of the video standard.

The behavior of *read()* when called during the active picture period or the vertical blanking separating the top and bottom field depends on the discarding policy. A driver discarding the oldest frames keeps capturing into an internal buffer, continuously overwriting the previously, not read frame, and returns the frame being received at the time of the *read()* call as soon as it is complete.

A driver discarding the newest frames stops capturing until the next *read()* call. The frame being received at *read()* time is discarded, returning the following frame instead. Again this implies a reduction of the capture rate to one half or less of the nominal frame rate. An example of this model is the video read mode of the bttv driver, initiating a DMA to user memory when *read()* is called and returning when the DMA finished.

In the multiple buffer model drivers maintain a ring of internal buffers, automatically advancing to the next free buffer. This allows continuous capturing when the application can empty the buffers fast enough. Again, the behavior when the driver runs out of free buffers depends on the discarding policy.

Applications can get and set the number of buffers used internally by the driver with the *VIDIOC_G_PARM* and *VIDIOC_S_PARM* ioctls. They are optional, however. The discarding policy is not reported and cannot be changed. For minimum requirements see *Interfaces* .

Return Value

On success, the number of bytes read is returned. It is not an error if this number is smaller than the number of bytes requested, or the amount of data required for one frame. This may happen for example because *read()* was interrupted by a signal. On error, -1 is returned, and the errno variable is set appropriately. In this case the next read will start at the beginning of a new frame. Possible error codes are:

EAGAIN Non-blocking I/O has been selected using O_NONBLOCK and no data was immediately available for reading.

EBADF fd is not a valid file descriptor or is not open for reading, or the process already has the maximum number of files open.

EBUSY The driver does not support multiple read streams and the device is already in use.

EFAULT buf references an inaccessible memory area.

EINTR The call was interrupted by a signal before any data was read.

EIO I/O error. This indicates some hardware problem or a failure to communicate with a remote device (USB camera etc.).

EINVAL The *read()* function is not supported by this driver, not on this device, or generally not on this type of device.

V4L2 select()

Name

v4l2-select - Synchronous I/O multiplexing

Synopsis

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
```

int **select**(int *nfds*, fd_set **readfds*, fd_set **writefds*, fd_set **exceptfds*, struct timeval **timeout*)

Arguments

nfds The highest-numbered file descriptor in any of the three sets, plus 1.

readfds File descriptions to be watched if a read() call won't block.

writefds File descriptions to be watched if a write() won't block.

exceptfds File descriptions to be watched for V4L2 events.

timeout Maximum time to wait.

Description

With the *select()* function applications can suspend execution until the driver has captured data or is ready to accept data for output.

When streaming I/O has been negotiated this function waits until a buffer has been filled or displayed and can be dequeued with the *VIDIOC_DQBUF* ioctl. When buffers are already in the outgoing queue of the driver the function returns immediately.

On success *select()* returns the total number of bits set in `struct fd_set()`. When the function timed out it returns a value of zero. On failure it returns -1 and the `errno` variable is set appropriately. When the application did not call *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* or *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* yet the *select()* function succeeds, setting the bit of the file descriptor in `readfds` or `writefds`, but subsequent *VIDIOC_DQBUF* calls will fail. [1]

When use of the *read()* function has been negotiated and the driver does not capture yet, the *select()* function starts capturing. When that fails, *select()* returns successful and a subsequent *read()* call, which also attempts to start capturing, will return an appropriate error code. When the driver captures continuously (as opposed to, for example, still images) and data is already available the *select()* function returns immediately.

When use of the *write()* function has been negotiated the *select()* function just waits until the driver is ready for a non-blocking *write()* call.

All drivers implementing the *read()* or *write()* function or streaming I/O must also support the *select()* function.

For more details see the *select()* manual page.

Return Value

On success, *select()* returns the number of descriptors contained in the three returned descriptor sets, which will be zero if the timeout expired. On error -1 is returned, and the `errno` variable is set appropriately; the sets and `timeout` are undefined. Possible error codes are:

EBADF One or more of the file descriptor sets specified a file descriptor that is not open.

EBUSY The driver does not support multiple read or write streams and the device is already in use.

EFAULT The `readfds`, `writefds`, `exceptfds` or `timeout` pointer references an inaccessible memory area.

[1] The Linux kernel implements *select()* like the *poll()* function, but *select()* cannot return a POLLERR.

EINTR The call was interrupted by a signal.

EINVAL The `nfds` argument is less than zero or greater than `FD_SETSIZE`.

V4L2 write()

Name

v4l2-write - Write to a V4L2 device

Synopsis

```
#include <unistd.h>
```

ssize_t **write**(int *fd*, void **buf*, size_t *count*)

Arguments

`fd` File descriptor returned by *open()* .

`buf` Buffer with data to be written

`count` Number of bytes at the buffer

Description

write() writes up to `count` bytes to the device referenced by the file descriptor `fd` from the buffer starting at `buf`. When the hardware outputs are not active yet, this function enables them. When `count` is zero, *write()* returns 0 without any other effect.

When the application does not provide more data in time, the previous video frame, raw VBI image, sliced VPS or WSS data is displayed again. Sliced Teletext or Closed Caption data is not repeated, the driver inserts a blank line instead.

Return Value

On success, the number of bytes written are returned. Zero indicates nothing was written. On error, -1 is returned, and the `errno` variable is set appropriately. In this case the next write will start at the beginning of a new frame. Possible error codes are:

EAGAIN Non-blocking I/O has been selected using the *O_NONBLOCK* flag and no buffer space was available to write the data immediately.

EBADF `fd` is not a valid file descriptor or is not open for writing.

EBUSY The driver does not support multiple write streams and the device is already in use.

EFAULT `buf` references an inaccessible memory area.

EINTR The call was interrupted by a signal before any data was written.

EIO I/O error. This indicates some hardware problem.

EINVAL The *write()* function is not supported by this driver, not on this device, or generally not on this type of device.

1.2.9 Common definitions for V4L2 and V4L2 subdev interfaces

Common selection definitions

While the *V4L2 selection API* and *V4L2 subdev selection APIs* are very similar, there's one fundamental difference between the two. On sub-device API, the selection rectangle refers to the media bus format, and is bound to a sub-device's pad. On the V4L2 interface the selection rectangles refer to the in-memory pixel format.

This section defines the common definitions of the selection interfaces on the two APIs.

Selection targets

The precise meaning of the selection targets may be dependent on which of the two interfaces they are used.

Table 1.159: Selection target definitions

Target name	id	Definition	Valid for V4L2	Valid for V4L2 subdev
V4L2_SEL_TGT_CROP	0x0000	Crop rectangle. Defines the cropped area.	Yes	Yes
V4L2_SEL_TGT_CROP_DEFAULT	0x0001	Suggested cropping rectangle that covers the "whole picture".	Yes	No
V4L2_SEL_TGT_CROP_BOUNDS	0x0002	Bounds of the crop rectangle. All valid crop rectangles fit inside the crop bounds rectangle.	Yes	Yes
V4L2_SEL_TGT_NATIVE_SIZE	0x0003	The native size of the device, e.g. a sensor's pixel array. `left` and `top` fields are zero for this target. Setting the native size will generally only make sense for memory to memory devices where the software can create a canvas of a given size in which for example a video frame can be composed. In that case V4L2_SEL_TGT_NATIVE_SIZE can be used to configure the size of that canvas.	Yes	Yes
V4L2_SEL_TGT_COMPOSE	0x0100	Compose rectangle. Used to configure scaling and composition.	Yes	Yes
V4L2_SEL_TGT_COMPOSE_DEFAULT	0x0101	Suggested composition rectangle that covers the "whole picture".	Yes	No
V4L2_SEL_TGT_COMPOSE_BOUNDS	0x0102	Bounds of the compose rectangle. All valid compose rectangles fit inside the compose bounds rectangle.	Yes	Yes
V4L2_SEL_TGT_COMPOSE_PADDED	0x0103	The active area and all padding pixels that are inserted or modified by hardware.	Yes	No

Selection flags

Table 1.160: Selection flag definitions

Flag name	id	Definition	Valid for V4L2	Valid for V4L2 subdev
V4L2_SEL_FLAG_GE	(1 << 0)	Suggest the driver it should choose greater or equal rectangle (in size) than was requested. Albeit the driver may choose a lesser size, it will only do so due to hardware limitations. Without this flag (and V4L2_SEL_FLAG_LE) the behaviour is to choose the closest possible rectangle.	Yes	Yes
V4L2_SEL_FLAG_LE	(1 << 1)	Suggest the driver it should choose lesser or equal rectangle (in size) than was requested. Albeit the driver may choose a greater size, it will only do so due to hardware limitations.	Yes	Yes
V4L2_SEL_FLAG_KEEP_CONFIG	(1 << 2)	The configuration must not be propagated to any further processing steps. If this flag is not given, the configuration is propagated inside the subdevice to all further processing steps.	No	Yes

1.2.10 Video For Linux Two Header File

videodev2.h

```
/*
 *  Video for Linux Two header file
 *
 *  Copyright (C) 1999-2012 the contributors
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  Alternatively you can redistribute this file under the terms of the
 *  BSD license as stated below:
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions
 *  are met:
 *  1. Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *  2. Redistributions in binary form must reproduce the above copyright
```

```
 *     notice, this list of conditions and the following disclaimer in
 *     the documentation and/or other materials provided with the
 *     distribution.
 *  3. The names of its contributors may not be used to endorse or promote
 *     products derived from this software without specific prior written
 *     permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 *  TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 *  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 *  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 *  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 *  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *      Header file for v4l or V4L2 drivers and applications
 * with public API.
 * All kernel-specific stuff were moved to media/v4l2-dev.h, so
 * no #if __KERNEL tests are allowed here
 *
 *      See https://linuxtv.org for more info
 *
 *      Author: Bill Dirks <bill@thedirks.org>
 *              Justin Schoeman
 *              Hans Verkuil <hverkuil@xs4all.nl>
 *              et al.
 */
#ifndef _UAPI__LINUX_VIDEODEV2_H
#define _UAPI__LINUX_VIDEODEV2_H

#ifndef __KERNEL__
#include <sys/time.h>
#endif
#include <linux/compiler.h>
#include <linux/ioctl.h>
#include <linux/types.h>
#include <linux/v4l2-common.h>
#include <linux/v4l2-controls.h>

/*
 * Common stuff for both V4L1 and V4L2
 * Moved from videodev.h
 */
#define VIDEO_MAX_FRAME               32
#define VIDEO_MAX_PLANES               8

/*
 *      M I S C E L L A N E O U S
 */

/*  Four-character-code (FOURCC) */
#define v4l2_fourcc(a, b, c, d)\
        ((__u32)(a) | ((__u32)(b) << 8) | ((__u32)(c) << 16) | ((__u32)(d) << 24))
#define v4l2_fourcc_be(a, b, c, d)      (v4l2_fourcc(a, b, c, d) | (1 << 31))
```

```
/*
 *      E N U M S
 */
enum  v4l2_field
 {
         V4L2_FIELD_ANY
           = 0, /* driver can choose from none,
                                          top, bottom, interlaced
                                          depending on whatever it thinks
                                          is approximate ... */
         V4L2_FIELD_NONE
          = 1, /* this device has no fields ... */
         V4L2_FIELD_TOP
           = 2, /* top field only */
         V4L2_FIELD_BOTTOM
        = 3, /* bottom field only */
         V4L2_FIELD_INTERLACED
    = 4, /* both fields interlaced */
         V4L2_FIELD_SEQ_TB
        = 5, /* both fields sequential into one
                                          buffer, top-bottom order */
         V4L2_FIELD_SEQ_BT
        = 6, /* same as above + bottom-top order */
         V4L2_FIELD_ALTERNATE
     = 7, /* both fields alternating into
                                          separate buffers */
         V4L2_FIELD_INTERLACED_TB
 = 8, /* both fields interlaced, top field
                                          first and the top field is
                                          transmitted first */
         V4L2_FIELD_INTERLACED_BT
 = 9, /* both fields interlaced, top field
                                          first and the bottom field is
                                          transmitted first */
};
#define V4L2_FIELD_HAS_TOP(field)       \
        ((field) ==  V4L2_FIELD_TOP
      ||\
         (field) ==  V4L2_FIELD_INTERLACED
 ||\
         (field) ==  V4L2_FIELD_INTERLACED_TB
 ||\
         (field) ==  V4L2_FIELD_INTERLACED_BT
 ||\
         (field) ==  V4L2_FIELD_SEQ_TB
   ||\
         (field) ==  V4L2_FIELD_SEQ_BT
)
#define V4L2_FIELD_HAS_BOTTOM(field)    \
        ((field) ==  V4L2_FIELD_BOTTOM
   ||\
         (field) ==  V4L2_FIELD_INTERLACED
 ||\
         (field) ==  V4L2_FIELD_INTERLACED_TB
 ||\
         (field) ==  V4L2_FIELD_INTERLACED_BT
 ||\
```

```
         (field) ==  V4L2_FIELD_SEQ_TB
   ||\
         (field) ==  V4L2_FIELD_SEQ_BT
)
#define V4L2_FIELD_HAS_BOTH(field)      \
        ((field) ==  V4L2_FIELD_INTERLACED
 ||\
         (field) ==  V4L2_FIELD_INTERLACED_TB
 ||\
         (field) ==  V4L2_FIELD_INTERLACED_BT
 ||\
         (field) ==  V4L2_FIELD_SEQ_TB
 ||\
         (field) ==  V4L2_FIELD_SEQ_BT
)
#define V4L2_FIELD_HAS_T_OR_B(field)    \
        ((field) ==  V4L2_FIELD_BOTTOM
 ||\
         (field) ==  V4L2_FIELD_TOP
 ||\
         (field) ==  V4L2_FIELD_ALTERNATE
)

enum  v4l2_buf_type
 {
         V4L2_BUF_TYPE_VIDEO_CAPTURE
        = 1,
         V4L2_BUF_TYPE_VIDEO_OUTPUT
         = 2,
         V4L2_BUF_TYPE_VIDEO_OVERLAY
        = 3,
         V4L2_BUF_TYPE_VBI_CAPTURE
          = 4,
         V4L2_BUF_TYPE_VBI_OUTPUT
           = 5,
         V4L2_BUF_TYPE_SLICED_VBI_CAPTURE
   = 6,
         V4L2_BUF_TYPE_SLICED_VBI_OUTPUT
    = 7,
         V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY
 = 8,
         V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE
 = 9,
         V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE
  = 10,
         V4L2_BUF_TYPE_SDR_CAPTURE
          = 11,
         V4L2_BUF_TYPE_SDR_OUTPUT
           = 12,
        /* Deprecated, do not use */
        V4L2_BUF_TYPE_PRIVATE              = 0x80,
};

#define V4L2_TYPE_IS_MULTIPLANAR(type)                  \
        ((type) ==  V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE
   \
         || (type) ==  V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE
)
```

```
#define V4L2_TYPE_IS_OUTPUT(type)                                       \
        ((type) ==  V4L2_BUF_TYPE_VIDEO_OUTPUT
                  \
         || (type) ==  V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE
          \
         || (type) ==  V4L2_BUF_TYPE_VIDEO_OVERLAY
              \
         || (type) ==  V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY
        \
         || (type) ==  V4L2_BUF_TYPE_VBI_OUTPUT
                  \
         || (type) ==  V4L2_BUF_TYPE_SLICED_VBI_OUTPUT
            \
         || (type) ==  V4L2_BUF_TYPE_SDR_OUTPUT
)

enum  v4l2_tuner_type
 {
         V4L2_TUNER_RADIO
             = 1,
         V4L2_TUNER_ANALOG_TV
         = 2,
        V4L2_TUNER_DIGITAL_TV        = 3,
         V4L2_TUNER_SDR
               = 4,
         V4L2_TUNER_RF
                = 5,
};

/* Deprecated, do not use */
#define V4L2_TUNER_ADC   V4L2_TUNER_SDR


enum  v4l2_memory
 {
         V4L2_MEMORY_MMAP
             = 1,
         V4L2_MEMORY_USERPTR
          = 2,
         V4L2_MEMORY_OVERLAY
          = 3,
         V4L2_MEMORY_DMABUF
           = 4,
};

/* see also http://vektor.theorem.ca/graphics/ycbcr/ */
enum  v4l2_colorspace
 {
        /*
         * Default colorspace, i.e. let the driver figure it out.
         * Can only be used with video capture.
         */
         V4L2_COLORSPACE_DEFAULT
       = 0,

        /* SMPTE 170M: used for broadcast NTSC/PAL SDTV */
         V4L2_COLORSPACE_SMPTE170M
```

```
    = 1,

       /* Obsolete pre-1998 SMPTE 240M HDTV standard, superseded by Rec 709 */
        V4L2_COLORSPACE_SMPTE240M
    = 2,

       /* Rec.709: used for HDTV */
        V4L2_COLORSPACE_REC709
       = 3,

       /*
        * Deprecated, do not use. No driver will ever return this. This was
        * based on a misunderstanding of the bt878 datasheet.
        */
       V4L2_COLORSPACE_BT878         = 4,

       /*
        * NTSC 1953 colorspace. This only makes sense when dealing with
        * really, really old NTSC recordings. Superseded by SMPTE 170M.
        */
        V4L2_COLORSPACE_470_SYSTEM_M
 = 5,

       /*
        * EBU Tech 3213 PAL/SECAM colorspace. This only makes sense when
        * dealing with really old PAL/SECAM recordings. Superseded by
        * SMPTE 170M.
        */
        V4L2_COLORSPACE_470_SYSTEM_BG
= 6,

       /*
        * Effectively shorthand for  V4L2_COLORSPACE_SRGB
,  V4L2_YCBCR_ENC_601

        * and V4L2_QUANTIZATION_FULL_RANGE. To be used for (Motion-)JPEG.
        */
        V4L2_COLORSPACE_JPEG
         = 7,

       /* For RGB colorspaces such as produces by most webcams. */
        V4L2_COLORSPACE_SRGB
         = 8,

       /* AdobeRGB colorspace */
        V4L2_COLORSPACE_ADOBERGB
     = 9,

       /* BT.2020 colorspace, used for UHDTV. */
        V4L2_COLORSPACE_BT2020
       = 10,

       /* Raw colorspace: for RAW unprocessed images */
        V4L2_COLORSPACE_RAW
          = 11,

       /* DCI-P3 colorspace, used by cinema projectors */
        V4L2_COLORSPACE_DCI_P3
```

```
        = 12,
};

/*
 * Determine how COLORSPACE_DEFAULT should map to a proper colorspace.
 * This depends on whether this is a SDTV image (use SMPTE 170M), an
 * HDTV image (use Rec. 709), or something else (use sRGB).
 */
#define V4L2_MAP_COLORSPACE_DEFAULT(is_sdtv, is_hdtv) \
        ((is_sdtv) ?  V4L2_COLORSPACE_SMPTE170M
 : \
         ((is_hdtv) ?  V4L2_COLORSPACE_REC709
 :  V4L2_COLORSPACE_SRGB
))

enum  v4l2_xfer_func
 {
        /*
         * Mapping of  V4L2_XFER_FUNC_DEFAULT
 to actual transfer functions
         * for the various colorspaces:
         *
         *  V4L2_COLORSPACE_SMPTE170M
,  V4L2_COLORSPACE_470_SYSTEM_M
,
         *  V4L2_COLORSPACE_470_SYSTEM_BG
,  V4L2_COLORSPACE_REC709
 and
         *  V4L2_COLORSPACE_BT2020
:  V4L2_XFER_FUNC_709

         *
         *  V4L2_COLORSPACE_SRGB
,  V4L2_COLORSPACE_JPEG
:  V4L2_XFER_FUNC_SRGB

         *
         *  V4L2_COLORSPACE_ADOBERGB
:  V4L2_XFER_FUNC_ADOBERGB

         *
         *  V4L2_COLORSPACE_SMPTE240M
:  V4L2_XFER_FUNC_SMPTE240M

         *
         *  V4L2_COLORSPACE_RAW
:  V4L2_XFER_FUNC_NONE

         *
         *  V4L2_COLORSPACE_DCI_P3
:  V4L2_XFER_FUNC_DCI_P3

         */
         V4L2_XFER_FUNC_DEFAULT
     = 0,
         V4L2_XFER_FUNC_709
         = 1,
         V4L2_XFER_FUNC_SRGB
```

```
        = 2,
         V4L2_XFER_FUNC_ADOBERGB
    = 3,
         V4L2_XFER_FUNC_SMPTE240M
   = 4,
         V4L2_XFER_FUNC_NONE
        = 5,
         V4L2_XFER_FUNC_DCI_P3
      = 6,
         V4L2_XFER_FUNC_SMPTE2084
   = 7,
};

/*
 * Determine how XFER_FUNC_DEFAULT should map to a proper transfer function.
 * This depends on the colorspace.
 */
#define V4L2_MAP_XFER_FUNC_DEFAULT(colsp) \
        ((colsp) ==  V4L2_COLORSPACE_ADOBERGB
 ?  V4L2_XFER_FUNC_ADOBERGB
 : \
         ((colsp) ==  V4L2_COLORSPACE_SMPTE240M
 ?  V4L2_XFER_FUNC_SMPTE240M
 : \
          ((colsp) ==  V4L2_COLORSPACE_DCI_P3
 ?  V4L2_XFER_FUNC_DCI_P3
 : \
           ((colsp) ==  V4L2_COLORSPACE_RAW
 ?  V4L2_XFER_FUNC_NONE
 : \
            ((colsp) ==  V4L2_COLORSPACE_SRGB
 || (colsp) ==  V4L2_COLORSPACE_JPEG
 ? \
              V4L2_XFER_FUNC_SRGB
 :  V4L2_XFER_FUNC_709
)))))

enum  v4l2_ycbcr_encoding
 {
        /*
         * Mapping of  V4L2_YCBCR_ENC_DEFAULT
 to actual encodings for the
         * various colorspaces:
         *
         *  V4L2_COLORSPACE_SMPTE170M
,  V4L2_COLORSPACE_470_SYSTEM_M
,
         *  V4L2_COLORSPACE_470_SYSTEM_BG
,  V4L2_COLORSPACE_SRGB
,
         *  V4L2_COLORSPACE_ADOBERGB
 and  V4L2_COLORSPACE_JPEG
:  V4L2_YCBCR_ENC_601

         *
         *  V4L2_COLORSPACE_REC709
 and  V4L2_COLORSPACE_DCI_P3
:  V4L2_YCBCR_ENC_709
```

```
         *
         *  V4L2_COLORSPACE_BT2020
:  V4L2_YCBCR_ENC_BT2020

         *
         *  V4L2_COLORSPACE_SMPTE240M
:  V4L2_YCBCR_ENC_SMPTE240M

         */
         V4L2_YCBCR_ENC_DEFAULT
        = 0,

        /* ITU-R 601 -- SDTV */
         V4L2_YCBCR_ENC_601
            = 1,

        /* Rec. 709 -- HDTV */
         V4L2_YCBCR_ENC_709
            = 2,

        /* ITU-R 601/EN 61966-2-4 Extended Gamut -- SDTV */
         V4L2_YCBCR_ENC_XV601
          = 3,

        /* Rec. 709/EN 61966-2-4 Extended Gamut -- HDTV */
         V4L2_YCBCR_ENC_XV709
          = 4,

#ifndef __KERNEL__
        /*
         * sYCC (Y'CbCr encoding of sRGB), identical to ENC_601. It was added
         * originally due to a misunderstanding of the sYCC standard. It should
         * not be used, instead use V4L2_YCBCR_ENC_601.
         */
         V4L2_YCBCR_ENC_SYCC
           = 5,
#endif

        /* BT.2020 Non-constant Luminance Y'CbCr */
         V4L2_YCBCR_ENC_BT2020
         = 6,

        /* BT.2020 Constant Luminance Y'CbcCrc */
         V4L2_YCBCR_ENC_BT2020_CONST_LUM
 = 7,

        /* SMPTE 240M -- Obsolete HDTV */
         V4L2_YCBCR_ENC_SMPTE240M
      = 8,
};

/*
 * enum  v4l2_hsv_encoding
 values should not collide with the ones from
 * enum v4l2_ycbcr_encoding.
 */
enum  v4l2_hsv_encoding
```

```
 {

        /* Hue mapped to 0 - 179 */
         V4L2_HSV_ENC_180
                = 128,

        /* Hue mapped to 0-255 */
         V4L2_HSV_ENC_256
                = 129,
};

/*
 * Determine how YCBCR_ENC_DEFAULT should map to a proper Y'CbCr encoding.
 * This depends on the colorspace.
 */
#define V4L2_MAP_YCBCR_ENC_DEFAULT(colsp) \
        (((colsp) ==  V4L2_COLORSPACE_REC709
 || \
          (colsp) ==  V4L2_COLORSPACE_DCI_P3
) ?  V4L2_YCBCR_ENC_709
 : \
         ((colsp) ==  V4L2_COLORSPACE_BT2020
 ?  V4L2_YCBCR_ENC_BT2020
 : \
          ((colsp) ==  V4L2_COLORSPACE_SMPTE240M
 ?  V4L2_YCBCR_ENC_SMPTE240M
 : \
            V4L2_YCBCR_ENC_601
)))

enum  v4l2_quantization
 {
        /*
         * The default for R'G'B' quantization is always full range, except
         * for the BT2020 colorspace. For Y'CbCr the quantization is always
         * limited range, except for COLORSPACE_JPEG, SRGB, ADOBERGB,
         * XV601 or XV709: those are full range.
         */
         V4L2_QUANTIZATION_DEFAULT
     = 0,
         V4L2_QUANTIZATION_FULL_RANGE
  = 1,
         V4L2_QUANTIZATION_LIM_RANGE
   = 2,
};

/*
 * Determine how QUANTIZATION_DEFAULT should map to a proper quantization.
 * This depends on whether the image is RGB or not, the colorspace and the
 * Y'CbCr encoding.
 */
#define V4L2_MAP_QUANTIZATION_DEFAULT(is_rgb_or_hsv, colsp, ycbcr_enc) \
        (((is_rgb_or_hsv) && (colsp) ==  V4L2_COLORSPACE_BT2020
) ? \
          V4L2_QUANTIZATION_LIM_RANGE
 : \
         (((is_rgb_or_hsv) || (ycbcr_enc) ==  V4L2_YCBCR_ENC_XV601
 || \
```

```
          (ycbcr_enc) ==  V4L2_YCBCR_ENC_XV709
 || (colsp) ==  V4L2_COLORSPACE_JPEG
) || \
          (colsp) ==  V4L2_COLORSPACE_ADOBERGB
 || (colsp) ==  V4L2_COLORSPACE_SRGB
 ? \
          V4L2_QUANTIZATION_FULL_RANGE
 :  V4L2_QUANTIZATION_LIM_RANGE
))

enum  v4l2_priority
 {
         V4L2_PRIORITY_UNSET
       = 0,  /* not initialized */
         V4L2_PRIORITY_BACKGROUND
  = 1,
         V4L2_PRIORITY_INTERACTIVE
 = 2,
         V4L2_PRIORITY_RECORD
      = 3,
         V4L2_PRIORITY_DEFAULT
     =  V4L2_PRIORITY_INTERACTIVE
,
};

struct  v4l2_rect
 {
        __s32   left;
        __s32   top;
        __u32   width;
        __u32   height;
};

struct  v4l2_fract
 {
        __u32   numerator;
        __u32   denominator;
};

/**
  * struct  v4l2_capability
 - Describes V4L2 device caps returned by  VIDIOC_QUERYCAP
  *
  * @driver:       name of the driver module (e.g. ``bttv'')
  * @card:         name of the card (e.g. ``Hauppauge WinTV'')
  * @bus_info:     name of the bus (e.g. ``PCI:'' + pci_name(pci_dev) )
  * @version:      KERNEL_VERSION
  * @capabilities: capabilities of the physical device as a whole
  * @device_caps:  capabilities accessed via this particular device (node)
  * @reserved:     reserved fields for future extensions
  */
struct  v4l2_capability
 {
        __u8    driver[16];
        __u8    card[32];
        __u8    bus_info[32];
        __u32   version;
        __u32   capabilities;
```

```
        __u32   device_caps;
        __u32   reserved[3];
};

/* Values for `capabilities' field */
#define  V4L2_CAP_VIDEO_CAPTURE          0x00000001  /* Is a video capture device */
#define  V4L2_CAP_VIDEO_OUTPUT           0x00000002  /* Is a video output device */
#define  V4L2_CAP_VIDEO_OVERLAY          0x00000004  /* Can do video overlay */
#define  V4L2_CAP_VBI_CAPTURE            0x00000010  /* Is a raw VBI capture device */
#define  V4L2_CAP_VBI_OUTPUT             0x00000020  /* Is a raw VBI output device */
#define  V4L2_CAP_SLICED_VBI_CAPTURE     0x00000040  /* Is a sliced VBI capture de-
vice */
#define  V4L2_CAP_SLICED_VBI_OUTPUT      0x00000080  /* Is a sliced VBI output de-
vice */
#define  V4L2_CAP_RDS_CAPTURE            0x00000100  /* RDS data capture */
#define  V4L2_CAP_VIDEO_OUTPUT_OVERLAY   0x00000200  /* Can do video output overlay */
#define  V4L2_CAP_HW_FREQ_SEEK           0x00000400  /* Can do hardware fre-
quency seek  */
#define  V4L2_CAP_RDS_OUTPUT             0x00000800  /* Is an RDS encoder */

/* Is a video capture device that supports multiplanar formats */
#define  V4L2_CAP_VIDEO_CAPTURE_MPLANE    0x00001000
/* Is a video output device that supports multiplanar formats */
#define  V4L2_CAP_VIDEO_OUTPUT_MPLANE     0x00002000
/* Is a video mem-to-mem device that supports multiplanar formats */
#define  V4L2_CAP_VIDEO_M2M_MPLANE        0x00004000
/* Is a video mem-to-mem device */
#define  V4L2_CAP_VIDEO_M2M               0x00008000

#define  V4L2_CAP_TUNER                   0x00010000  /* has a tuner */
#define  V4L2_CAP_AUDIO                   0x00020000  /* has audio support */
#define  V4L2_CAP_RADIO                   0x00040000  /* is a radio device */
#define  V4L2_CAP_MODULATOR               0x00080000  /* has a modulator */

#define  V4L2_CAP_SDR_CAPTURE             0x00100000  /* Is a SDR capture device */
#define  V4L2_CAP_EXT_PIX_FORMAT          0x00200000  /* Supports the ex-
tended pixel format */
#define  V4L2_CAP_SDR_OUTPUT              0x00400000  /* Is a SDR output device */

#define  V4L2_CAP_READWRITE               0x01000000  /* read/write systemcalls */
#define  V4L2_CAP_ASYNCIO                 0x02000000  /* async I/O */
#define  V4L2_CAP_STREAMING               0x04000000  /* streaming I/O ioctls */

#define  V4L2_CAP_TOUCH                   0x10000000  /* Is a touch device */

#define  V4L2_CAP_DEVICE_CAPS             0x80000000  /* sets device capabili-
ties field */

/*
 *      V I D E O   I M A G E   F O R M A T
 */
struct  v4l2_pix_format
 {
        __u32                   width;
        __u32                   height;
        __u32                   pixelformat;
        __u32                   field;          /* enum  v4l2_field
 */
```

```
        __u32                   bytesperline;   /* for padding, zero if unused */
        __u32                   sizeimage;
        __u32                   colorspace;     /* enum  v4l2_colorspace
 */
        __u32                   priv;           /* private data, depends on pixelfor-
mat */
        __u32                   flags;          /* format flags (V4L2_PIX_FMT_FLAG_*) */
        union {
                /* enum  v4l2_ycbcr_encoding
 */
                __u32                   ycbcr_enc;
                /* enum  v4l2_hsv_encoding
 */
                __u32                   hsv_enc;
        };
        __u32                   quantization;   /* enum  v4l2_quantization
 */
        __u32                   xfer_func;      /* enum  v4l2_xfer_func
 */
};

/*      Pixel format         FOURCC                         depth  Description  */

/* RGB formats */
#define  V4L2_PIX_FMT_RGB332   v4l2_fourcc(`R', `G', `B', `1') /*  8  RGB-3-3-2     */
#define  V4L2_PIX_FMT_RGB444   v4l2_fourcc(`R', `4', `4', `4') /* 16  xxxxrrrr gggg-
bbbb */
#define  V4L2_PIX_FMT_ARGB444  v4l2_fourcc(`A', `R', `1', `2') /* 16  aaaarrrr gggg-
bbbb */
#define  V4L2_PIX_FMT_XRGB444  v4l2_fourcc(`X', `R', `1', `2') /* 16  xxxxrrrr gggg-
bbbb */
#define  V4L2_PIX_FMT_RGB555   v4l2_fourcc(`R', `G', `B', `O') /* 16  RGB-5-5-5     */
#define  V4L2_PIX_FMT_ARGB555  v4l2_fourcc(`A', `R', `1', `5') /* 16  ARGB-1-5-5-5  */
#define  V4L2_PIX_FMT_XRGB555  v4l2_fourcc(`X', `R', `1', `5') /* 16  XRGB-1-5-5-5  */
#define  V4L2_PIX_FMT_RGB565   v4l2_fourcc(`R', `G', `B', `P') /* 16  RGB-5-6-5     */
#define  V4L2_PIX_FMT_RGB555X  v4l2_fourcc(`R', `G', `B', `Q') /* 16  RGB-5-5-5 BE  */
#define  V4L2_PIX_FMT_ARGB555X  v4l2_fourcc_be(`A', `R', `1', `5') /* 16  ARGB-5-5-
5 BE */
#define  V4L2_PIX_FMT_XRGB555X  v4l2_fourcc_be(`X', `R', `1', `5') /* 16  XRGB-5-5-
5 BE */
#define  V4L2_PIX_FMT_RGB565X  v4l2_fourcc(`R', `G', `B', `R') /* 16  RGB-5-6-5 BE  */
#define  V4L2_PIX_FMT_BGR666   v4l2_fourcc(`B', `G', `R', `H') /* 18  BGR-6-6-6     */
#define  V4L2_PIX_FMT_BGR24    v4l2_fourcc(`B', `G', `R', `3') /* 24  BGR-8-8-8     */
#define  V4L2_PIX_FMT_RGB24    v4l2_fourcc(`R', `G', `B', `3') /* 24  RGB-8-8-8     */
#define  V4L2_PIX_FMT_BGR32    v4l2_fourcc(`B', `G', `R', `4') /* 32  BGR-8-8-8-8   */
#define  V4L2_PIX_FMT_ABGR32   v4l2_fourcc(`A', `R', `2', `4') /* 32  BGRA-8-8-8-8  */
#define  V4L2_PIX_FMT_XBGR32   v4l2_fourcc(`X', `R', `2', `4') /* 32  BGRX-8-8-8-8  */
#define  V4L2_PIX_FMT_RGB32    v4l2_fourcc(`R', `G', `B', `4') /* 32  RGB-8-8-8-8   */
#define  V4L2_PIX_FMT_ARGB32   v4l2_fourcc(`B', `A', `2', `4') /* 32  ARGB-8-8-8-8  */
#define  V4L2_PIX_FMT_XRGB32   v4l2_fourcc(`B', `X', `2', `4') /* 32  XRGB-8-8-8-8  */

/* Grey formats */
#define  V4L2_PIX_FMT_GREY     v4l2_fourcc(`G', `R', `E', `Y') /*  8  Greyscale     */
#define  V4L2_PIX_FMT_Y4       v4l2_fourcc(`Y', `0', `4', ` `) /*  4  Greyscale     */
#define  V4L2_PIX_FMT_Y6       v4l2_fourcc(`Y', `0', `6', ` `) /*  6  Greyscale     */
#define  V4L2_PIX_FMT_Y10      v4l2_fourcc(`Y', `1', `0', ` `) /* 10  Greyscale     */
#define  V4L2_PIX_FMT_Y12      v4l2_fourcc(`Y', `1', `2', ` `) /* 12  Greyscale     */
#define  V4L2_PIX_FMT_Y16      v4l2_fourcc(`Y', `1', `6', ` `) /* 16  Greyscale     */
```

```
#define  V4L2_PIX_FMT_Y16_BE   v4l2_fourcc_be(`Y', `1', `6', ` `) /* 16  Greyscale BE  */

/* Grey bit-packed formats */
#define  V4L2_PIX_FMT_Y10BPACK     v4l2_fourcc(`Y', `1', `0', `B') /* 10  Greyscale bit-
packed */

/* Palette formats */
#define  V4L2_PIX_FMT_PAL8     v4l2_fourcc(`P', `A', `L', `8') /*  8  8-bit palette */

/* Chrominance formats */
#define  V4L2_PIX_FMT_UV8      v4l2_fourcc(`U', `V', `8', ` `) /*  8  UV 4:4 */

/* Luminance+Chrominance formats */
#define  V4L2_PIX_FMT_YUYV     v4l2_fourcc(`Y', `U', `Y', `V') /* 16  YUV 4:2:2     */
#define  V4L2_PIX_FMT_YYUV     v4l2_fourcc(`Y', `Y', `U', `V') /* 16  YUV 4:2:2     */
#define  V4L2_PIX_FMT_YVYU     v4l2_fourcc(`Y', `V', `Y', `U') /* 16 YVU 4:2:2 */
#define  V4L2_PIX_FMT_UYVY     v4l2_fourcc(`U', `Y', `V', `Y') /* 16  YUV 4:2:2     */
#define  V4L2_PIX_FMT_VYUY     v4l2_fourcc(`V', `Y', `U', `Y') /* 16  YUV 4:2:2     */
#define  V4L2_PIX_FMT_Y41P     v4l2_fourcc(`Y', `4', `1', `P') /* 12  YUV 4:1:1     */
#define  V4L2_PIX_FMT_YUV444   v4l2_fourcc(`Y', `4', `4', `4') /* 16  xxxxyyyy uuu-
uvvvv */
#define  V4L2_PIX_FMT_YUV555   v4l2_fourcc(`Y', `U', `V', `O') /* 16  YUV-5-5-5     */
#define  V4L2_PIX_FMT_YUV565   v4l2_fourcc(`Y', `U', `V', `P') /* 16  YUV-5-6-5     */
#define  V4L2_PIX_FMT_YUV32    v4l2_fourcc(`Y', `U', `V', `4') /* 32  YUV-8-8-8-8   */
#define  V4L2_PIX_FMT_HI240    v4l2_fourcc(`H', `I', `2', `4') /*  8  8-bit color   */
#define  V4L2_PIX_FMT_HM12     v4l2_fourcc(`H', `M', `1', `2') /*  8  YUV 4:2:0 16x16 mac-
roblocks */
#define  V4L2_PIX_FMT_M420     v4l2_fourcc(`M', `4', `2', `0') /* 12  YUV 4:2:0 2 lines y, 1
terleaved */

/* two planes -- one Y, one Cr + Cb interleaved  */
#define  V4L2_PIX_FMT_NV12     v4l2_fourcc(`N', `V', `1', `2') /* 12  Y/CbCr 4:2:0  */
#define  V4L2_PIX_FMT_NV21     v4l2_fourcc(`N', `V', `2', `1') /* 12  Y/CrCb 4:2:0  */
#define  V4L2_PIX_FMT_NV16     v4l2_fourcc(`N', `V', `1', `6') /* 16  Y/CbCr 4:2:2  */
#define  V4L2_PIX_FMT_NV61     v4l2_fourcc(`N', `V', `6', `1') /* 16  Y/CrCb 4:2:2  */
#define  V4L2_PIX_FMT_NV24     v4l2_fourcc(`N', `V', `2', `4') /* 24  Y/CbCr 4:4:4  */
#define  V4L2_PIX_FMT_NV42     v4l2_fourcc(`N', `V', `4', `2') /* 24  Y/CrCb 4:4:4  */

/* two non contiguous planes - one Y, one Cr + Cb interleaved  */
#define  V4L2_PIX_FMT_NV12M    v4l2_fourcc(`N', `M', `1', `2') /* 12  Y/CbCr 4:2:0  */
#define  V4L2_PIX_FMT_NV21M    v4l2_fourcc(`N', `M', `2', `1') /* 21  Y/CrCb 4:2:0  */
#define  V4L2_PIX_FMT_NV16M    v4l2_fourcc(`N', `M', `1', `6') /* 16  Y/CbCr 4:2:2  */
#define  V4L2_PIX_FMT_NV61M    v4l2_fourcc(`N', `M', `6', `1') /* 16  Y/CrCb 4:2:2  */
#define  V4L2_PIX_FMT_NV12MT   v4l2_fourcc(`T', `M', `1', `2') /* 12  Y/CbCr 4:2:0 64x32 mac-
roblocks */
#define  V4L2_PIX_FMT_NV12MT_16X16  v4l2_fourcc(`V', `M', `1', `2') /* 12  Y/CbCr 4:2:0 16x16
roblocks */

/* three planes - Y Cb, Cr */
#define  V4L2_PIX_FMT_YUV410   v4l2_fourcc(`Y', `U', `V', `9') /*  9  YUV 4:1:0     */
#define  V4L2_PIX_FMT_YVU410   v4l2_fourcc(`Y', `V', `U', `9') /*  9  YVU 4:1:0     */
#define  V4L2_PIX_FMT_YUV411P  v4l2_fourcc(`4', `1', `1', `P') /* 12  YVU411 planar */
#define  V4L2_PIX_FMT_YUV420   v4l2_fourcc(`Y', `U', `1', `2') /* 12  YUV 4:2:0     */
#define  V4L2_PIX_FMT_YVU420   v4l2_fourcc(`Y', `V', `1', `2') /* 12  YVU 4:2:0     */
#define  V4L2_PIX_FMT_YUV422P  v4l2_fourcc(`4', `2', `2', `P') /* 16  YVU422 planar */

/* three non contiguous planes - Y, Cb, Cr */
#define  V4L2_PIX_FMT_YUV420M  v4l2_fourcc(`Y', `M', `1', `2') /* 12  YUV420 planar */
```

```
#define  V4L2_PIX_FMT_YVU420M  v4l2_fourcc(`Y', `M', `2', `1') /* 12  YVU420 planar */
#define  V4L2_PIX_FMT_YUV422M  v4l2_fourcc(`Y', `M', `1', `6') /* 16  YUV422 planar */
#define  V4L2_PIX_FMT_YVU422M  v4l2_fourcc(`Y', `M', `6', `1') /* 16  YVU422 planar */
#define  V4L2_PIX_FMT_YUV444M  v4l2_fourcc(`Y', `M', `2', `4') /* 24  YUV444 planar */
#define  V4L2_PIX_FMT_YVU444M  v4l2_fourcc(`Y', `M', `4', `2') /* 24  YVU444 planar */

/* Bayer formats - see http://www.siliconimaging.com/RGB%20Bayer.htm */
#define  V4L2_PIX_FMT_SBGGR8   v4l2_fourcc(`B', `A', `8', `1') /*  8  BGBG.. GRGR.. */
#define  V4L2_PIX_FMT_SGBRG8   v4l2_fourcc(`G', `B', `R', `G') /*  8  GBGB.. RGRG.. */
#define  V4L2_PIX_FMT_SGRBG8   v4l2_fourcc(`G', `R', `B', `G') /*  8  GRGR.. BGBG.. */
#define  V4L2_PIX_FMT_SRGGB8   v4l2_fourcc(`R', `G', `G', `B') /*  8  RGRG.. GBGB.. */
#define  V4L2_PIX_FMT_SBGGR10  v4l2_fourcc(`B', `G', `1', `0') /* 10  BGBG.. GRGR.. */
#define  V4L2_PIX_FMT_SGBRG10  v4l2_fourcc(`G', `B', `1', `0') /* 10  GBGB.. RGRG.. */
#define  V4L2_PIX_FMT_SGRBG10  v4l2_fourcc(`B', `A', `1', `0') /* 10  GRGR.. BGBG.. */
#define  V4L2_PIX_FMT_SRGGB10  v4l2_fourcc(`R', `G', `1', `0') /* 10  RGRG.. GBGB.. */
        /* 10bit raw bayer packed, 5 bytes for every 4 pixels */
#define  V4L2_PIX_FMT_SBGGR10P  v4l2_fourcc(`p', `B', `A', `A')
#define  V4L2_PIX_FMT_SGBRG10P  v4l2_fourcc(`p', `G', `A', `A')
#define  V4L2_PIX_FMT_SGRBG10P  v4l2_fourcc(`p', `g', `A', `A')
#define  V4L2_PIX_FMT_SRGGB10P  v4l2_fourcc(`p', `R', `A', `A')
        /* 10bit raw bayer a-law compressed to 8 bits */
#define  V4L2_PIX_FMT_SBGGR10ALAW8  v4l2_fourcc(`a', `B', `A', `8')
#define  V4L2_PIX_FMT_SGBRG10ALAW8  v4l2_fourcc(`a', `G', `A', `8')
#define  V4L2_PIX_FMT_SGRBG10ALAW8  v4l2_fourcc(`a', `g', `A', `8')
#define  V4L2_PIX_FMT_SRGGB10ALAW8  v4l2_fourcc(`a', `R', `A', `8')
        /* 10bit raw bayer DPCM compressed to 8 bits */
#define  V4L2_PIX_FMT_SBGGR10DPCM8  v4l2_fourcc(`b', `B', `A', `8')
#define  V4L2_PIX_FMT_SGBRG10DPCM8  v4l2_fourcc(`b', `G', `A', `8')
#define  V4L2_PIX_FMT_SGRBG10DPCM8  v4l2_fourcc(`B', `D', `1', `0')
#define  V4L2_PIX_FMT_SRGGB10DPCM8  v4l2_fourcc(`b', `R', `A', `8')
#define  V4L2_PIX_FMT_SBGGR12  v4l2_fourcc(`B', `G', `1', `2') /* 12  BGBG.. GRGR.. */
#define  V4L2_PIX_FMT_SGBRG12  v4l2_fourcc(`G', `B', `1', `2') /* 12  GBGB.. RGRG.. */
#define  V4L2_PIX_FMT_SGRBG12  v4l2_fourcc(`B', `A', `1', `2') /* 12  GRGR.. BGBG.. */
#define  V4L2_PIX_FMT_SRGGB12  v4l2_fourcc(`R', `G', `1', `2') /* 12  RGRG.. GBGB.. */
#define  V4L2_PIX_FMT_SBGGR16  v4l2_fourcc(`B', `Y', `R', `2') /* 16  BGBG.. GRGR.. */
#define  V4L2_PIX_FMT_SGBRG16  v4l2_fourcc(`G', `B', `1', `6') /* 16  GBGB.. RGRG.. */
#define  V4L2_PIX_FMT_SGRBG16  v4l2_fourcc(`G', `R', `1', `6') /* 16  GRGR.. BGBG.. */
#define  V4L2_PIX_FMT_SRGGB16  v4l2_fourcc(`R', `G', `1', `6') /* 16  RGRG.. GBGB.. */

/* HSV formats */
#define  V4L2_PIX_FMT_HSV24  v4l2_fourcc(`H', `S', `V', `3')
#define  V4L2_PIX_FMT_HSV32  v4l2_fourcc(`H', `S', `V', `4')

/* compressed formats */
#define  V4L2_PIX_FMT_MJPEG     v4l2_fourcc(`M', `J', `P', `G') /* Motion-JPEG   */
#define  V4L2_PIX_FMT_JPEG      v4l2_fourcc(`J', `P', `E', `G') /* JFIF JPEG     */
#define  V4L2_PIX_FMT_DV        v4l2_fourcc(`d', `v', `s', `d') /* 1394          */
#define  V4L2_PIX_FMT_MPEG      v4l2_fourcc(`M', `P', `E', `G') /* MPEG-1/2/4 Multi-
plexed */
#define  V4L2_PIX_FMT_H264      v4l2_fourcc(`H', `2', `6', `4') /* H264 with start codes */
#define  V4L2_PIX_FMT_H264_NO_SC  v4l2_fourcc(`A', `V', `C', `1') /* H264 with-
out start codes */
#define  V4L2_PIX_FMT_H264_MVC  v4l2_fourcc(`M', `2', `6', `4') /* H264 MVC */
#define  V4L2_PIX_FMT_H263      v4l2_fourcc(`H', `2', `6', `3') /* H263          */
#define  V4L2_PIX_FMT_MPEG1     v4l2_fourcc(`M', `P', `G', `1') /* MPEG-1 ES     */
#define  V4L2_PIX_FMT_MPEG2     v4l2_fourcc(`M', `P', `G', `2') /* MPEG-2 ES     */
#define  V4L2_PIX_FMT_MPEG4     v4l2_fourcc(`M', `P', `G', `4') /* MPEG-4 part 2 ES */
#define  V4L2_PIX_FMT_XVID      v4l2_fourcc(`X', `V', `I', `D') /* Xvid           */
```

```
#define  V4L2_PIX_FMT_VC1_ANNEX_G  v4l2_fourcc(`V', `C', `1', `G') /* SMPTE 421M An-
nex G compliant stream */
#define  V4L2_PIX_FMT_VC1_ANNEX_L  v4l2_fourcc(`V', `C', `1', `L') /* SMPTE 421M An-
nex L compliant stream */
#define  V4L2_PIX_FMT_VP8       v4l2_fourcc(`V', `P', `8', `0') /* VP8 */
#define  V4L2_PIX_FMT_VP9       v4l2_fourcc(`V', `P', `9', `0') /* VP9 */

/*  Vendor-specific formats   */
#define  V4L2_PIX_FMT_CPIA1     v4l2_fourcc(`C', `P', `I', `A') /* cpia1 YUV */
#define  V4L2_PIX_FMT_WNVA      v4l2_fourcc(`W', `N', `V', `A') /* Winnov hw com-
press */
#define  V4L2_PIX_FMT_SN9C10X   v4l2_fourcc(`S', `9', `1', `0') /* SN9C10x compres-
sion */
#define  V4L2_PIX_FMT_SN9C20X_I420  v4l2_fourcc(`S', `9', `2', `0') /* SN9C20x YUV 4:2:0 */
#define  V4L2_PIX_FMT_PWC1      v4l2_fourcc(`P', `W', `C', `1') /* pwc older webcam */
#define  V4L2_PIX_FMT_PWC2      v4l2_fourcc(`P', `W', `C', `2') /* pwc newer webcam */
#define  V4L2_PIX_FMT_ET61X251  v4l2_fourcc(`E', `6', `2', `5') /* ET61X251 compres-
sion */
#define  V4L2_PIX_FMT_SPCA501   v4l2_fourcc(`S', `5', `0', `1') /* YUYV per line */
#define  V4L2_PIX_FMT_SPCA505   v4l2_fourcc(`S', `5', `0', `5') /* YYUV per line */
#define  V4L2_PIX_FMT_SPCA508   v4l2_fourcc(`S', `5', `0', `8') /* YUVY per line */
#define  V4L2_PIX_FMT_SPCA561   v4l2_fourcc(`S', `5', `6', `1') /* com-
pressed GBRG bayer */
#define  V4L2_PIX_FMT_PAC207    v4l2_fourcc(`P', `2', `0', `7') /* com-
pressed BGGR bayer */
#define  V4L2_PIX_FMT_MR97310A  v4l2_fourcc(`M', `3', `1', `0') /* com-
pressed BGGR bayer */
#define  V4L2_PIX_FMT_JL2005BCD  v4l2_fourcc(`J', `L', `2', `0') /* com-
pressed RGGB bayer */
#define  V4L2_PIX_FMT_SN9C2028  v4l2_fourcc(`S', `O', `N', `X') /* com-
pressed GBRG bayer */
#define  V4L2_PIX_FMT_SQ905C    v4l2_fourcc(`9', `0', `5', `C') /* com-
pressed RGGB bayer */
#define  V4L2_PIX_FMT_PJPG      v4l2_fourcc(`P', `J', `P', `G') /* Pixart 73xx JPEG */
#define  V4L2_PIX_FMT_OV511     v4l2_fourcc(`O', `5', `1', `1') /* ov511 JPEG */
#define  V4L2_PIX_FMT_OV518     v4l2_fourcc(`O', `5', `1', `8') /* ov518 JPEG */
#define  V4L2_PIX_FMT_STV0680   v4l2_fourcc(`S', `6', `8', `0') /* stv0680 bayer */
#define  V4L2_PIX_FMT_TM6000    v4l2_fourcc(`T', `M', `6', `0') /* tm5600/tm60x0 */
#define  V4L2_PIX_FMT_CIT_YYVYUY  v4l2_fourcc(`C', `I', `T', `V') /* one line of Y then 1 lin
#define  V4L2_PIX_FMT_KONICA420   v4l2_fourcc(`K', `O', `N', `I') /* YUV420 pla-
nar in blocks of 256 pixels */
#define  V4L2_PIX_FMT_JPGL        v4l2_fourcc(`J', `P', `G', `L') /* JPEG-Lite */
#define  V4L2_PIX_FMT_SE401       v4l2_fourcc(`S', `4', `0', `1') /* se401 janggu com-
pressed rgb */
#define  V4L2_PIX_FMT_S5C_UYVY_JPG  v4l2_fourcc(`S', `5', `C', `I') /* S5C73M3 inter-
leaved UYVY/JPEG */
#define  V4L2_PIX_FMT_Y8I       v4l2_fourcc(`Y', `8', `I', ` `) /* Greyscale 8-
bit L/R interleaved */
#define  V4L2_PIX_FMT_Y12I      v4l2_fourcc(`Y', `1', `2', `I') /* Greyscale 12-
bit L/R interleaved */
#define  V4L2_PIX_FMT_Z16       v4l2_fourcc(`Z', `1', `6', ` `) /* Depth data 16-bit */
#define  V4L2_PIX_FMT_MT21C     v4l2_fourcc(`M', `T', `2', `1') /* Mediatek com-
pressed block mode  */

/* SDR formats - used only for Software Defined Radio devices */
#define  V4L2_SDR_FMT_CU8           v4l2_fourcc(`C', `U', `0', `8') /* IQ u8 */
#define  V4L2_SDR_FMT_CU16LE        v4l2_fourcc(`C', `U', `1', `6') /* IQ u16le */
#define  V4L2_SDR_FMT_CS8           v4l2_fourcc(`C', `S', `0', `8') /* complex s8 */
```

```
#define  V4L2_SDR_FMT_CS14LE        v4l2_fourcc(`C', `S', `1', `4') /* complex s14le */
#define  V4L2_SDR_FMT_RU12LE        v4l2_fourcc(`R', `U', `1', `2') /* real u12le */

/* Touch formats - used for Touch devices */
#define  V4L2_TCH_FMT_DELTA_TD16  v4l2_fourcc(`T', `D', `1', `6') /* 16-
bit signed deltas */
#define  V4L2_TCH_FMT_DELTA_TD08  v4l2_fourcc(`T', `D', `0', `8') /* 8-
bit signed deltas */
#define  V4L2_TCH_FMT_TU16        v4l2_fourcc(`T', `U', `1', `6') /* 16-bit un-
signed touch data */
#define  V4L2_TCH_FMT_TU08        v4l2_fourcc(`T', `U', `0', `8') /* 8-bit un-
signed touch data */

/* priv field value to indicates that subsequent fields are valid. */
#define  V4L2_PIX_FMT_PRIV_MAGIC
         0xfeedcafe

/* Flags */
#define  V4L2_PIX_FMT_FLAG_PREMUL_ALPHA   0x00000001

/*
 *      F O R M A T   E N U M E R A T I O N
 */
struct  v4l2_fmtdesc
 {
        __u32               index;             /* Format number      */
        __u32               type;              /* enum  v4l2_buf_type
 */
        __u32               flags;
        __u8                description[32];   /* Description string */
        __u32               pixelformat;       /* Format fourcc      */
        __u32               reserved[4];
};

#define  V4L2_FMT_FLAG_COMPRESSED  0x0001
#define  V4L2_FMT_FLAG_EMULATED    0x0002

        /* Frame Size and frame rate enumeration */
/*
 *      F R A M E   S I Z E   E N U M E R A T I O N
 */
enum  v4l2_frmsizetypes
 {
         V4L2_FRMSIZE_TYPE_DISCRETE
      = 1,
         V4L2_FRMSIZE_TYPE_CONTINUOUS
    = 2,
         V4L2_FRMSIZE_TYPE_STEPWISE
      = 3,
};

struct  v4l2_frmsize_discrete
 {
        __u32                   width;          /* Frame width [pixel] */
        __u32                   height;         /* Frame height [pixel] */
};

struct  v4l2_frmsize_stepwise
```

```
 {
        __u32                   min_width;      /* Minimum frame width [pixel] */
        __u32                   max_width;      /* Maximum frame width [pixel] */
        __u32                   step_width;     /* Frame width step size [pixel] */
        __u32                   min_height;     /* Minimum frame height [pixel] */
        __u32                   max_height;     /* Maximum frame height [pixel] */
        __u32                   step_height;    /* Frame height step size [pixel] */
};

struct  v4l2_frmsizeenum
 {
        __u32                   index;          /* Frame size number */
        __u32                   pixel_format;   /* Pixel format */
        __u32                   type;           /* Frame size type the device sup-
ports. */

        union {                                 /* Frame size */
                struct  v4l2_frmsize_discrete
    discrete;
                struct  v4l2_frmsize_stepwise
    stepwise;
        };

        __u32   reserved[2];                    /* Reserved space for future use */
};

/*
 *      F R A M E   R A T E   E N U M E R A T I O N
 */
enum  v4l2_frmivaltypes
 {
         V4L2_FRMIVAL_TYPE_DISCRETE
      = 1,
         V4L2_FRMIVAL_TYPE_CONTINUOUS
    = 2,
         V4L2_FRMIVAL_TYPE_STEPWISE
      = 3,
};

struct  v4l2_frmival_stepwise
 {
        struct  v4l2_fract
       min;             /* Minimum frame interval [s] */
        struct  v4l2_fract
       max;             /* Maximum frame interval [s] */
        struct  v4l2_fract
       step;            /* Frame interval step size [s] */
};

struct  v4l2_frmivalenum
 {
        __u32                   index;          /* Frame format index */
        __u32                   pixel_format;   /* Pixel format */
        __u32                   width;          /* Frame width */
        __u32                   height;         /* Frame height */
        __u32                   type;           /* Frame interval type the device sup-
ports. */
```

```
        union {                                 /* Frame interval */
                struct  v4l2_fract
               discrete;
                struct  v4l2_frmival_stepwise
    stepwise;
        };

        __u32   reserved[2];                    /* Reserved space for future use */
};

/*
 *      T I M E C O D E
 */
struct  v4l2_timecode
 {
        __u32   type;
        __u32   flags;
        __u8    frames;
        __u8    seconds;
        __u8    minutes;
        __u8    hours;
        __u8    userbits[4];
};

/*  Type  */
#define  V4L2_TC_TYPE_24FPS             1
#define  V4L2_TC_TYPE_25FPS             2
#define  V4L2_TC_TYPE_30FPS             3
#define  V4L2_TC_TYPE_50FPS             4
#define  V4L2_TC_TYPE_60FPS             5

/*  Flags  */
#define  V4L2_TC_FLAG_DROPFRAME         0x0001 /* ``drop-frame'' mode */
#define  V4L2_TC_FLAG_COLORFRAME        0x0002
#define  V4L2_TC_USERBITS_field         0x000C
#define  V4L2_TC_USERBITS_USERDEFINED   0x0000
#define  V4L2_TC_USERBITS_8BITCHARS     0x0008
/* The above is based on SMPTE timecodes */

struct  v4l2_jpegcompression
 {
        int quality;

        int  APPn;              /* Number of APP segment to be written,
                                 * must be 0..15 */
        int  APP_len;           /* Length of data in JPEG APPn segment */
        char APP_data[60];      /* Data in the JPEG APPn segment. */

        int  COM_len;           /* Length of data in JPEG COM segment */
        char COM_data[60];      /* Data in JPEG COM segment */

        __u32 jpeg_markers;     /* Which markers should go into the JPEG
                                 * output. Unless you exactly know what
                                 * you do, leave them untouched.
                                 * Including less markers will make the
                                 * resulting code smaller, but there will
                                 * be fewer applications which can read it.
                                 * The presence of the APP and COM marker
```

```
                                * is influenced by APP_len and COM_len
                                * ONLY, not by this property! */

#define  V4L2_JPEG_MARKER_DHT  (1<<3)    /* Define Huffman Tables */
#define  V4L2_JPEG_MARKER_DQT  (1<<4)    /* Define Quantization Tables */
#define  V4L2_JPEG_MARKER_DRI  (1<<5)    /* Define Restart Interval */
#define  V4L2_JPEG_MARKER_COM  (1<<6)    /* Comment segment */
#define  V4L2_JPEG_MARKER_APP  (1<<7)    /* App segment, driver will
                                         * always use APP0 */
};

/*
 *      M E M O R Y - M A P P I N G   B U F F E R S
 */
struct  v4l2_requestbuffers
 {
        __u32                   count;
        __u32                   type;           /* enum  v4l2_buf_type
 */
        __u32                   memory;         /* enum  v4l2_memory
 */
        __u32                   reserved[2];
};

/**
 * struct  v4l2_plane
 - plane info for multi-planar buffers
 * @bytesused:          number of bytes occupied by data in the plane (payload)
 * @length:            size of this plane (NOT the payload) in bytes
 * @mem_offset:        when memory in the associated struct  v4l2_buffer
 is
 *                      V4L2_MEMORY_MMAP
, equals the offset from the start of
 *                      the device memory for this plane (or is a ``cookie'' that
 *                      should be passed to mmap() called on the video node)
 * @userptr:           when memory is  V4L2_MEMORY_USERPTR
, a userspace pointer
 *                      pointing to this plane
 * @fd:                when memory is  V4L2_MEMORY_DMABUF
, a userspace file
 *                      descriptor associated with this plane
 * @data_offset:       offset in the plane to the start of data; usually 0,
 *                      unless there is a header in front of the data
 *
 * Multi-planar buffers consist of one or more planes, e.g. an YCbCr buffer
 * with two planes can have one plane for Y, and another for interleaved CbCr
 * components. Each plane can reside in a separate memory buffer, or even in
 * a completely separate memory node (e.g. in embedded devices).
 */
struct  v4l2_plane
 {
        __u32                   bytesused;
        __u32                   length;
        union {
                __u32           mem_offset;
                unsigned long   userptr;
                __s32           fd;
        } m;
```

```
        __u32                   data_offset;
        __u32                   reserved[11];
};

/**
 * struct  v4l2_buffer
 - video buffer info
 * @index:      id number of the buffer
 * @type:       enum  v4l2_buf_type
; buffer type (type == *_MPLANE for
 *              multiplanar buffers);
 * @bytesused:  number of bytes occupied by data in the buffer (payload);
 *              unused (set to 0) for multiplanar buffers
 * @flags:      buffer informational flags
 * @field:      enum  v4l2_field
; field order of the image in the buffer
 * @timestamp:  frame timestamp
 * @timecode:   frame timecode
 * @sequence:   sequence count of this frame
 * @memory:     enum  v4l2_memory
; the method, in which the actual video data is
 *              passed
 * @offset:     for non-multiplanar buffers with memory ==  V4L2_MEMORY_MMAP
;
 *              offset from the start of the device memory for this plane,
 *              (or a ``cookie'' that should be passed to mmap() as offset)
 * @userptr:    for non-multiplanar buffers with memory ==  V4L2_MEMORY_USERPTR
;
 *              a userspace pointer pointing to this buffer
 * @fd:         for non-multiplanar buffers with memory ==  V4L2_MEMORY_DMABUF
;
 *              a userspace file descriptor associated with this buffer
 * @planes:     for multiplanar buffers; userspace pointer to the array of plane
 *              info structs for this buffer
 * @length:     size in bytes of the buffer (NOT its payload) for single-plane
 *              buffers (when type != *_MPLANE); number of elements in the
 *              planes array for multi-plane buffers
 *
 * Contains data exchanged by application and driver using one of the Streaming
 * I/O methods.
 */
struct  v4l2_buffer
 {
        __u32                   index;
        __u32                   type;
        __u32                   bytesused;
        __u32                   flags;
        __u32                   field;
        struct timeval          timestamp;
        struct  v4l2_timecode
    timecode;
        __u32                   sequence;

        /* memory location */
        __u32                   memory;
        union {
                __u32           offset;
                unsigned long   userptr;
```

```
                struct  v4l2_plane
 *planes;
                __s32           fd;
        } m;
        __u32                   length;
        __u32                   reserved2;
        __u32                   reserved;
};

/*  Flags for `flags' field */
/* Buffer is mapped (flag) */
#define  V4L2_BUF_FLAG_MAPPED                   0x00000001
/* Buffer is queued for processing */
#define  V4L2_BUF_FLAG_QUEUED                   0x00000002
/* Buffer is ready */
#define  V4L2_BUF_FLAG_DONE                     0x00000004
/* Image is a keyframe (I-frame) */
#define  V4L2_BUF_FLAG_KEYFRAME                 0x00000008
/* Image is a P-frame */
#define  V4L2_BUF_FLAG_PFRAME                   0x00000010
/* Image is a B-frame */
#define  V4L2_BUF_FLAG_BFRAME                   0x00000020
/* Buffer is ready, but the data contained within is corrupted. */
#define  V4L2_BUF_FLAG_ERROR                    0x00000040
/* timecode field is valid */
#define  V4L2_BUF_FLAG_TIMECODE                 0x00000100
/* Buffer is prepared for queuing */
#define  V4L2_BUF_FLAG_PREPARED                 0x00000400
/* Cache handling flags */
#define  V4L2_BUF_FLAG_NO_CACHE_INVALIDATE      0x00000800
#define  V4L2_BUF_FLAG_NO_CACHE_CLEAN           0x00001000
/* Timestamp type */
#define  V4L2_BUF_FLAG_TIMESTAMP_MASK           0x0000e000
#define  V4L2_BUF_FLAG_TIMESTAMP_UNKNOWN        0x00000000
#define  V4L2_BUF_FLAG_TIMESTAMP_MONOTONIC      0x00002000
#define  V4L2_BUF_FLAG_TIMESTAMP_COPY           0x00004000
/* Timestamp sources. */
#define  V4L2_BUF_FLAG_TSTAMP_SRC_MASK          0x00070000
#define  V4L2_BUF_FLAG_TSTAMP_SRC_EOF           0x00000000
#define  V4L2_BUF_FLAG_TSTAMP_SRC_SOE           0x00010000
/* mem2mem encoder/decoder */
#define  V4L2_BUF_FLAG_LAST                     0x00100000

/**
 * struct  v4l2_exportbuffer
 - export of video buffer as DMABUF file descriptor
 *
 * @index:      id number of the buffer
 * @type:       enum  v4l2_buf_type
; buffer type (type == *_MPLANE for
 *              multiplanar buffers);
 * @plane:      index of the plane to be exported, 0 for single plane queues
 * @flags:      flags for newly created file, currently only O_CLOEXEC is
 *              supported, refer to manual of open syscall for more details
 * @fd:         file descriptor associated with DMABUF (set by driver)
 *
 * Contains data used for exporting a video buffer as DMABUF file descriptor.
 * The buffer is identified by a `cookie' returned by  VIDIOC_QUERYBUF
```

```
 * (identical to the cookie used to mmap() the buffer to userspace). All
 * reserved fields must be set to zero. The field reserved0 is expected to
 * become a structure `type' allowing an alternative layout of the structure
 * content. Therefore this field should not be used for any other extensions.
 */
struct  v4l2_exportbuffer
 {
        __u32           type; /* enum  v4l2_buf_type
 */
        __u32           index;
        __u32           plane;
        __u32           flags;
        __s32           fd;
        __u32           reserved[11];
};

/*
 *      O V E R L A Y   P R E V I E W
 */
struct  v4l2_framebuffer
 {
        __u32                   capability;
        __u32                   flags;
/* FIXME: in theory we should pass something like PCI device + memory
 * region + offset instead of some physical address */
        void                    *base;
        struct {
                __u32           width;
                __u32           height;
                __u32           pixelformat;
                __u32           field;          /* enum  v4l2_field
 */
                __u32           bytesperline;   /* for padding, zero if unused */
                __u32           sizeimage;
                __u32           colorspace;     /* enum  v4l2_colorspace
 */
                __u32           priv;           /* reserved field, set to 0 */
        } fmt;
};
/*  Flags for the `capability' field. Read only */
#define  V4L2_FBUF_CAP_EXTERNOVERLAY      0x0001
#define  V4L2_FBUF_CAP_CHROMAKEY          0x0002
#define  V4L2_FBUF_CAP_LIST_CLIPPING      0x0004
#define  V4L2_FBUF_CAP_BITMAP_CLIPPING    0x0008
#define  V4L2_FBUF_CAP_LOCAL_ALPHA        0x0010
#define  V4L2_FBUF_CAP_GLOBAL_ALPHA       0x0020
#define  V4L2_FBUF_CAP_LOCAL_INV_ALPHA    0x0040
#define  V4L2_FBUF_CAP_SRC_CHROMAKEY      0x0080
/*  Flags for the `flags' field. */
#define  V4L2_FBUF_FLAG_PRIMARY           0x0001
#define  V4L2_FBUF_FLAG_OVERLAY           0x0002
#define  V4L2_FBUF_FLAG_CHROMAKEY         0x0004
#define  V4L2_FBUF_FLAG_LOCAL_ALPHA       0x0008
#define  V4L2_FBUF_FLAG_GLOBAL_ALPHA      0x0010
#define  V4L2_FBUF_FLAG_LOCAL_INV_ALPHA   0x0020
#define  V4L2_FBUF_FLAG_SRC_CHROMAKEY     0x0040

struct  v4l2_clip
```

```
 {
        struct  v4l2_rect
        c;
        struct  v4l2_clip
        __user *next;
};

struct  v4l2_window
 {
        struct  v4l2_rect
        w;
        __u32                   field;   /* enum  v4l2_field
 */
        __u32                   chromakey;
        struct  v4l2_clip
        __user *clips;
        __u32                   clipcount;
        void                    __user *bitmap;
        __u8                    global_alpha;
};

/*
 *      C A P T U R E   P A R A M E T E R S
 */
struct  v4l2_captureparm
 {
        __u32              capability;    /*  Supported modes */
        __u32              capturemode;   /*  Current mode */
        struct  v4l2_fract
  timeperframe;  /*  Time per frame in seconds */
        __u32              extendedmode;  /*  Driver-specific extensions */
        __u32              readbuffers;   /*  # of buffers for read */
        __u32              reserved[4];
};

/*  Flags for `capability' and `capturemode' fields */
#define  V4L2_MODE_HIGHQUALITY    0x0001  /*  High quality imaging mode */
#define  V4L2_CAP_TIMEPERFRAME
   0x1000  /*  timeperframe field is supported */

struct  v4l2_outputparm
 {
        __u32              capability;   /*  Supported modes */
        __u32              outputmode;   /*  Current mode */
        struct  v4l2_fract
  timeperframe; /*  Time per frame in seconds */
        __u32              extendedmode; /*  Driver-specific extensions */
        __u32              writebuffers; /*  # of buffers for write */
        __u32              reserved[4];
};

/*
 *      I N P U T   I M A G E   C R O P P I N G
 */
struct  v4l2_cropcap
 {
        __u32                   type;   /* enum  v4l2_buf_type
 */
```

```
        struct  v4l2_rect
        bounds;
        struct  v4l2_rect
        defrect;
        struct  v4l2_fract
       pixelaspect;
};

struct  v4l2_crop
 {
        __u32                   type;   /* enum  v4l2_buf_type
 */
        struct  v4l2_rect
        c;
};

/**
 * struct  v4l2_selection
 - selection info
 * @type:       buffer type (do not use *_MPLANE types)
 * @target:     Selection target, used to choose one of possible rectangles;
 *              defined in v4l2-common.h; V4L2_SEL_TGT_* .
 * @flags:      constraints flags, defined in v4l2-common.h; V4L2_SEL_FLAG_*.
 * @r:          coordinates of selection window
 * @reserved:   for future use, rounds structure size to 64 bytes, set to zero
 *
 * Hardware may use multiple helper windows to process a video stream.
 * The structure is used to exchange this selection areas between
 * an application and a driver.
 */
struct  v4l2_selection
 {
        __u32                   type;
        __u32                   target;
        __u32                   flags;
        struct  v4l2_rect
        r;
        __u32                   reserved[9];
};

/*
 *      A N A L O G   V I D E O   S T A N D A R D
 */

typedef __u64 v4l2_std_id;

/* one bit for each */
#define  V4L2_STD_PAL_B          ((v4l2_std_id)0x00000001)
#define  V4L2_STD_PAL_B1         ((v4l2_std_id)0x00000002)
#define  V4L2_STD_PAL_G          ((v4l2_std_id)0x00000004)
#define  V4L2_STD_PAL_H          ((v4l2_std_id)0x00000008)
#define  V4L2_STD_PAL_I          ((v4l2_std_id)0x00000010)
#define  V4L2_STD_PAL_D          ((v4l2_std_id)0x00000020)
#define  V4L2_STD_PAL_D1         ((v4l2_std_id)0x00000040)
#define  V4L2_STD_PAL_K          ((v4l2_std_id)0x00000080)

#define  V4L2_STD_PAL_M          ((v4l2_std_id)0x00000100)
#define  V4L2_STD_PAL_N          ((v4l2_std_id)0x00000200)
```

```
#define  V4L2_STD_PAL_Nc          ((v4l2_std_id)0x00000400)
#define  V4L2_STD_PAL_60          ((v4l2_std_id)0x00000800)

#define  V4L2_STD_NTSC_M          ((v4l2_std_id)0x00001000)       /* BTSC */
#define  V4L2_STD_NTSC_M_JP       ((v4l2_std_id)0x00002000)       /* EIA-J */
#define  V4L2_STD_NTSC_443        ((v4l2_std_id)0x00004000)
#define  V4L2_STD_NTSC_M_KR       ((v4l2_std_id)0x00008000)       /* FM A2 */

#define  V4L2_STD_SECAM_B         ((v4l2_std_id)0x00010000)
#define  V4L2_STD_SECAM_D         ((v4l2_std_id)0x00020000)
#define  V4L2_STD_SECAM_G         ((v4l2_std_id)0x00040000)
#define  V4L2_STD_SECAM_H         ((v4l2_std_id)0x00080000)
#define  V4L2_STD_SECAM_K         ((v4l2_std_id)0x00100000)
#define  V4L2_STD_SECAM_K1        ((v4l2_std_id)0x00200000)
#define  V4L2_STD_SECAM_L         ((v4l2_std_id)0x00400000)
#define  V4L2_STD_SECAM_LC        ((v4l2_std_id)0x00800000)

/* ATSC/HDTV */
#define  V4L2_STD_ATSC_8_VSB      ((v4l2_std_id)0x01000000)
#define  V4L2_STD_ATSC_16_VSB     ((v4l2_std_id)0x02000000)

/* FIXME:
   Although std_id is 64 bits, there is an issue on PPC32 architecture that
   makes switch(__u64) to break. So, there's a hack on v4l2-common.c rounding
   this value to 32 bits.
   As, currently, the max value is for  V4L2_STD_ATSC_16_VSB  (30 bits wide),
   it should work fine. However, if needed to add more than two standards,
   v4l2-common.c should be fixed.
 */

/*
 * Some macros to merge video standards in order to make live easier for the
 * drivers and V4L2 applications
 */

/*
 * ``Common'' NTSC/M - It should be noticed that  V4L2_STD_NTSC_443  is
 * Missing here.
 */
#define  V4L2_STD_NTSC            ( V4L2_STD_NTSC_M         |\
                                   V4L2_STD_NTSC_M_JP      |\
                                   V4L2_STD_NTSC_M_KR )
/* Secam macros */
#define  V4L2_STD_SECAM_DK        ( V4L2_STD_SECAM_D        |\
                                   V4L2_STD_SECAM_K        |\
                                   V4L2_STD_SECAM_K1 )
/* All Secam Standards */
#define  V4L2_STD_SECAM           ( V4L2_STD_SECAM_B        |\
                                   V4L2_STD_SECAM_G        |\
                                   V4L2_STD_SECAM_H        |\
                                   V4L2_STD_SECAM_DK       |\
                                   V4L2_STD_SECAM_L        |\
                                   V4L2_STD_SECAM_LC )
/* PAL macros */
#define  V4L2_STD_PAL_BG          ( V4L2_STD_PAL_B          |\
                                   V4L2_STD_PAL_B1         |\
                                   V4L2_STD_PAL_G )
#define  V4L2_STD_PAL_DK          ( V4L2_STD_PAL_D          |\
```

```
                                V4L2_STD_PAL_D1         |\
                                V4L2_STD_PAL_K )
/*
 * ``Common'' PAL - This macro is there to be compatible with the old
 * V4L1 concept of ``PAL'': /BGDKHI.
 * Several PAL standards are missing here: /M, /N and /Nc
 */
#define  V4L2_STD_PAL           ( V4L2_STD_PAL_BG         |\
                                V4L2_STD_PAL_DK         |\
                                V4L2_STD_PAL_H          |\
                                V4L2_STD_PAL_I )
/* Chroma ``agnostic'' standards */
#define  V4L2_STD_B             ( V4L2_STD_PAL_B          |\
                                V4L2_STD_PAL_B1         |\
                                V4L2_STD_SECAM_B )
#define  V4L2_STD_G             ( V4L2_STD_PAL_G          |\
                                V4L2_STD_SECAM_G )
#define  V4L2_STD_H             ( V4L2_STD_PAL_H          |\
                                V4L2_STD_SECAM_H )
#define  V4L2_STD_L             ( V4L2_STD_SECAM_L        |\
                                V4L2_STD_SECAM_LC )
#define  V4L2_STD_GH            ( V4L2_STD_G              |\
                                V4L2_STD_H )
#define  V4L2_STD_DK            ( V4L2_STD_PAL_DK         |\
                                V4L2_STD_SECAM_DK )
#define  V4L2_STD_BG            ( V4L2_STD_B              |\
                                V4L2_STD_G )
#define  V4L2_STD_MN            ( V4L2_STD_PAL_M          |\
                                V4L2_STD_PAL_N          |\
                                V4L2_STD_PAL_Nc         |\
                                V4L2_STD_NTSC )

/* Standards where MTS/BTSC stereo could be found */
#define  V4L2_STD_MTS           ( V4L2_STD_NTSC_M         |\
                                V4L2_STD_PAL_M          |\
                                V4L2_STD_PAL_N          |\
                                V4L2_STD_PAL_Nc )

/* Standards for Countries with 60Hz Line frequency */
#define  V4L2_STD_525_60        ( V4L2_STD_PAL_M          |\
                                V4L2_STD_PAL_60         |\
                                V4L2_STD_NTSC           |\
                                V4L2_STD_NTSC_443 )
/* Standards for Countries with 50Hz Line frequency */
#define  V4L2_STD_625_50        ( V4L2_STD_PAL            |\
                                V4L2_STD_PAL_N          |\
                                V4L2_STD_PAL_Nc         |\
                                V4L2_STD_SECAM )

#define  V4L2_STD_ATSC          ( V4L2_STD_ATSC_8_VSB     |\
                                V4L2_STD_ATSC_16_VSB )
/* Macros with none and all analog standards */
#define  V4L2_STD_UNKNOWN       0
#define  V4L2_STD_ALL           ( V4L2_STD_525_60         |\
                                V4L2_STD_625_50 )

struct  v4l2_standard
 {
```

```
        __u32                index;
        v4l2_std_id          id;
        __u8                 name[24];
        struct  v4l2_fract
    frameperiod; /* Frames, not fields */
        __u32                framelines;
        __u32                reserved[4];
};

/*
 *      D V     B T     T I M I N G S
 */

/** struct  v4l2_bt_timings
 - BT.656/BT.1120 timing data
 * @width:      total width of the active video in pixels
 * @height:     total height of the active video in lines
 * @interlaced: Interlaced or progressive
 * @polarities: Positive or negative polarities
 * @pixelclock: Pixel clock in HZ. Ex. 74.25MHz->74250000
 * @hfrontporch:Horizontal front porch in pixels
 * @hsync:      Horizontal Sync length in pixels
 * @hbackporch: Horizontal back porch in pixels
 * @vfrontporch:Vertical front porch in lines
 * @vsync:      Vertical Sync length in lines
 * @vbackporch: Vertical back porch in lines
 * @il_vfrontporch:Vertical front porch for the even field
 *              (aka field 2) of interlaced field formats
 * @il_vsync:   Vertical Sync length for the even field
 *              (aka field 2) of interlaced field formats
 * @il_vbackporch:Vertical back porch for the even field
 *              (aka field 2) of interlaced field formats
 * @standards:  Standards the timing belongs to
 * @flags:      Flags
 * @picture_aspect: The picture aspect ratio (hor/vert).
 * @cea861_vic: VIC code as per the CEA-861 standard.
 * @hdmi_vic:   VIC code as per the HDMI standard.
 * @reserved:   Reserved fields, must be zeroed.
 *
 * A note regarding vertical interlaced timings: height refers to the total
 * height of the active video frame (= two fields). The blanking timings refer
 * to the blanking of each field. So the height of the total frame is
 * calculated as follows:
 *
 * tot_height = height + vfrontporch + vsync + vbackporch +
 *                       il_vfrontporch + il_vsync + il_vbackporch
 *
 * The active height of each field is height / 2.
 */
struct  v4l2_bt_timings
 {
        __u32   width;
        __u32   height;
        __u32   interlaced;
        __u32   polarities;
        __u64   pixelclock;
        __u32   hfrontporch;
        __u32   hsync;
```

```
        __u32   hbackporch;
        __u32   vfrontporch;
        __u32   vsync;
        __u32   vbackporch;
        __u32   il_vfrontporch;
        __u32   il_vsync;
        __u32   il_vbackporch;
        __u32   standards;
        __u32   flags;
        struct  v4l2_fract
 picture_aspect;
        __u8    cea861_vic;
        __u8    hdmi_vic;
        __u8    reserved[46];
} __attribute__ ((packed));

/* Interlaced or progressive format */
#define  V4L2_DV_PROGRESSIVE
     0
#define  V4L2_DV_INTERLACED
      1

/* Polarities. If bit is not set, it is assumed to be negative polarity */
#define  V4L2_DV_VSYNC_POS_POL
   0x00000001
#define  V4L2_DV_HSYNC_POS_POL
   0x00000002

/* Timings standards */
#define  V4L2_DV_BT_STD_CEA861    (1 << 0)  /* CEA-861 Digital TV Profile */
#define  V4L2_DV_BT_STD_DMT       (1 << 1)  /* VESA Discrete Monitor Timings */
#define  V4L2_DV_BT_STD_CVT       (1 << 2)  /* VESA Coordinated Video Timings */
#define  V4L2_DV_BT_STD_GTF       (1 << 3)  /* VESA Generalized Timings Formula */
#define  V4L2_DV_BT_STD_SDI       (1 << 4)  /* SDI Timings */

/* Flags */

/*
 * CVT/GTF specific: timing uses reduced blanking (CVT) or the `Secondary
 * GTF' curve (GTF). In both cases the horizontal and/or vertical blanking
 * intervals are reduced, allowing a higher resolution over the same
 * bandwidth. This is a read-only flag.
 */
#define  V4L2_DV_FL_REDUCED_BLANKING              (1 << 0)
/*
 * CEA-861 specific: set for CEA-861 formats with a framerate of a multiple
 * of six. These formats can be optionally played at 1 / 1.001 speed.
 * This is a read-only flag.
 */
#define  V4L2_DV_FL_CAN_REDUCE_FPS                (1 << 1)
/*
 * CEA-861 specific: only valid for video transmitters, the flag is cleared
 * by receivers.
 * If the framerate of the format is a multiple of six, then the pixelclock
 * used to set up the transmitter is divided by 1.001 to make it compatible
 * with 60 Hz based standards such as NTSC and PAL-M that use a framerate of
 * 29.97 Hz. Otherwise this flag is cleared. If the transmitter can't generate
 * such frequencies, then the flag will also be cleared.
```

```
 */
#define  V4L2_DV_FL_REDUCED_FPS                  (1 << 2)
/*
 * Specific to interlaced formats: if set, then field 1 is really one half-line
 * longer and field 2 is really one half-line shorter, so each field has
 * exactly the same number of half-lines. Whether half-lines can be detected
 * or used depends on the hardware.
 */
#define  V4L2_DV_FL_HALF_LINE                    (1 << 3)
/*
 * If set, then this is a Consumer Electronics (CE) video format. Such formats
 * differ from other formats (commonly called IT formats) in that if RGB
 * encoding is used then by default the RGB values use limited range (i.e.
 * use the range 16-235) as opposed to 0-255. All formats defined in CEA-861
 * except for the 640x480 format are CE formats.
 */
#define  V4L2_DV_FL_IS_CE_VIDEO                  (1 << 4)
/* Some formats like SMPTE-125M have an interlaced signal with a odd
 * total height. For these formats, if this flag is set, the first
 * field has the extra line. If not, it is the second field.
 */
#define  V4L2_DV_FL_FIRST_FIELD_EXTRA_LINE       (1 << 5)
/*
 * If set, then the picture_aspect field is valid. Otherwise assume that the
 * pixels are square, so the picture aspect ratio is the same as the width to
 * height ratio.
 */
#define  V4L2_DV_FL_HAS_PICTURE_ASPECT           (1 << 6)
/*
 * If set, then the cea861_vic field is valid and contains the Video
 * Identification Code as per the CEA-861 standard.
 */
#define  V4L2_DV_FL_HAS_CEA861_VIC               (1 << 7)
/*
 * If set, then the hdmi_vic field is valid and contains the Video
 * Identification Code as per the HDMI standard (HDMI Vendor Specific
 * InfoFrame).
 */
#define  V4L2_DV_FL_HAS_HDMI_VIC                 (1 << 8)

/* A few useful defines to calculate the total blanking and frame sizes */
#define V4L2_DV_BT_BLANKING_WIDTH(bt) \
        ((bt)->hfrontporch + (bt)->hsync + (bt)->hbackporch)
#define V4L2_DV_BT_FRAME_WIDTH(bt) \
        ((bt)->width + V4L2_DV_BT_BLANKING_WIDTH(bt))
#define V4L2_DV_BT_BLANKING_HEIGHT(bt) \
        ((bt)->vfrontporch + (bt)->vsync + (bt)->vbackporch + \
         (bt)->il_vfrontporch + (bt)->il_vsync + (bt)->il_vbackporch)
#define V4L2_DV_BT_FRAME_HEIGHT(bt) \
        ((bt)->height + V4L2_DV_BT_BLANKING_HEIGHT(bt))

/** struct  v4l2_dv_timings
 - DV timings
 * @type:       the type of the timings
 * @bt: BT656/1120 timings
 */
struct  v4l2_dv_timings
 {
```

```
        __u32 type;
        union {
                struct  v4l2_bt_timings
  bt;
                __u32   reserved[32];
        };
} __attribute__ ((packed));

/* Values for the type field */
#define  V4L2_DV_BT_656_1120      0       /* BT.656/1120 timing type */

/** struct  v4l2_enum_dv_timings
 - DV timings enumeration
 * @index:      enumeration index
 * @pad:        the pad number for which to enumerate timings (used with
 *              v4l-subdev nodes only)
 * @reserved:   must be zeroed
 * @timings:    the timings for the given index
 */
struct  v4l2_enum_dv_timings
 {
        __u32 index;
        __u32 pad;
        __u32 reserved[2];
        struct  v4l2_dv_timings
 timings;
};

/** struct  v4l2_bt_timings_cap
 - BT.656/BT.1120 timing capabilities
 * @min_width:          width in pixels
 * @max_width:          width in pixels
 * @min_height:         height in lines
 * @max_height:         height in lines
 * @min_pixelclock:     Pixel clock in HZ. Ex. 74.25MHz->74250000
 * @max_pixelclock:     Pixel clock in HZ. Ex. 74.25MHz->74250000
 * @standards:          Supported standards
 * @capabilities:       Supported capabilities
 * @reserved:           Must be zeroed
 */
struct  v4l2_bt_timings_cap
 {
        __u32   min_width;
        __u32   max_width;
        __u32   min_height;
        __u32   max_height;
        __u64   min_pixelclock;
        __u64   max_pixelclock;
        __u32   standards;
        __u32   capabilities;
        __u32   reserved[16];
} __attribute__ ((packed));

/* Supports interlaced formats */
#define  V4L2_DV_BT_CAP_INTERLACED        (1 << 0)
/* Supports progressive formats */
#define  V4L2_DV_BT_CAP_PROGRESSIVE       (1 << 1)
/* Supports CVT/GTF reduced blanking */
```

```
#define  V4L2_DV_BT_CAP_REDUCED_BLANKING  (1 << 2)
/* Supports custom formats */
#define  V4L2_DV_BT_CAP_CUSTOM            (1 << 3)

/** struct  v4l2_dv_timings_cap
 - DV timings capabilities
 * @type:       the type of the timings (same as in struct  v4l2_dv_timings
)
 * @pad:        the pad number for which to query capabilities (used with
 *              v4l-subdev nodes only)
 * @bt:         the BT656/1120 timings capabilities
 */
struct  v4l2_dv_timings_cap
 {
        __u32 type;
        __u32 pad;
        __u32 reserved[2];
        union {
                struct  v4l2_bt_timings_cap
 bt;
                __u32 raw_data[32];
        };
};

/*
 *      V I D E O   I N P U T S
 */
struct  v4l2_input
 {
        __u32        index;             /*  Which input */
        __u8         name[32];          /*  Label */
        __u32        type;              /*  Type of input */
        __u32        audioset;          /*  Associated audios (bitfield) */
        __u32        tuner;             /*  enum  v4l2_tuner_type
 */
        v4l2_std_id  std;
        __u32        status;
        __u32        capabilities;
        __u32        reserved[3];
};

/*  Values for the `type' field */
#define  V4L2_INPUT_TYPE_TUNER           1
#define  V4L2_INPUT_TYPE_CAMERA          2
#define  V4L2_INPUT_TYPE_TOUCH           3

/* field `status' - general */
#define  V4L2_IN_ST_NO_POWER     0x00000001  /* Attached device is off */
#define  V4L2_IN_ST_NO_SIGNAL    0x00000002
#define  V4L2_IN_ST_NO_COLOR     0x00000004

/* field `status' - sensor orientation */
/* If sensor is mounted upside down set both bits */
#define  V4L2_IN_ST_HFLIP        0x00000010 /* Frames are flipped horizontally */
#define  V4L2_IN_ST_VFLIP        0x00000020 /* Frames are flipped vertically */

/* field `status' - analog */
#define  V4L2_IN_ST_NO_H_LOCK    0x00000100  /* No horizontal sync lock */
```

```
#define  V4L2_IN_ST_COLOR_KILL   0x00000200  /* Color killer is active */
#define  V4L2_IN_ST_NO_V_LOCK    0x00000400  /* No vertical sync lock */
#define  V4L2_IN_ST_NO_STD_LOCK  0x00000800  /* No standard format lock */

/* field `status' - digital */
#define  V4L2_IN_ST_NO_SYNC      0x00010000  /* No synchronization lock */
#define  V4L2_IN_ST_NO_EQU       0x00020000  /* No equalizer lock */
#define  V4L2_IN_ST_NO_CARRIER   0x00040000  /* Carrier recovery failed */

/* field `status' - VCR and set-top box */
#define  V4L2_IN_ST_MACROVISION  0x01000000  /* Macrovision detected */
#define  V4L2_IN_ST_NO_ACCESS    0x02000000  /* Conditional access denied */
#define  V4L2_IN_ST_VTR          0x04000000  /* VTR time constant */

/* capabilities flags */
#define  V4L2_IN_CAP_DV_TIMINGS           0x00000002 /* Supports S_DV_TIMINGS */
#define V4L2_IN_CAP_CUSTOM_TIMINGS       V4L2_IN_CAP_DV_TIMINGS  /* For compatibil-
ity */
#define  V4L2_IN_CAP_STD                  0x00000004 /* Supports S_STD */
#define  V4L2_IN_CAP_NATIVE_SIZE          0x00000008 /* Supports setting native size */

/*
 *      V I D E O   O U T P U T S
 */
struct  v4l2_output
 {
        __u32        index;             /*  Which output */
        __u8         name[32];          /*  Label */
        __u32        type;              /*  Type of output */
        __u32        audioset;          /*  Associated audios (bitfield) */
        __u32        modulator;         /*  Associated modulator */
        v4l2_std_id  std;
        __u32        capabilities;
        __u32        reserved[3];
};
/*  Values for the `type' field */
#define  V4L2_OUTPUT_TYPE_MODULATOR              1
#define  V4L2_OUTPUT_TYPE_ANALOG                 2
#define  V4L2_OUTPUT_TYPE_ANALOGVGAOVERLAY       3

/* capabilities flags */
#define  V4L2_OUT_CAP_DV_TIMINGS          0x00000002 /* Supports S_DV_TIMINGS */
#define V4L2_OUT_CAP_CUSTOM_TIMINGS      V4L2_OUT_CAP_DV_TIMINGS  /* For compatibil-
ity */
#define  V4L2_OUT_CAP_STD                 0x00000004 /* Supports S_STD */
#define  V4L2_OUT_CAP_NATIVE_SIZE         0x00000008 /* Supports setting native size */

/*
 *      C O N T R O L S
 */
struct  v4l2_control
 {
        __u32                id;
        __s32                value;
};

struct  v4l2_ext_control
 {
```

```
        __u32 id;
        __u32 size;
        __u32 reserved2[1];
        union {
                __s32 value;
                __s64 value64;
                char __user *string;
                __u8 __user *p_u8;
                __u16 __user *p_u16;
                __u32 __user *p_u32;
                void __user *ptr;
        };
} __attribute__ ((packed));

struct  v4l2_ext_controls
 {
        union {
#ifndef __KERNEL__
                __u32 ctrl_class;
#endif
                __u32 which;
        };
        __u32 count;
        __u32 error_idx;
        __u32 reserved[2];
        struct  v4l2_ext_control
 *controls;
};

#define V4L2_CTRL_ID_MASK         (0x0fffffff)
#ifndef __KERNEL__
#define V4L2_CTRL_ID2CLASS(id)    ((id) & 0x0fff0000UL)
#endif
#define V4L2_CTRL_ID2WHICH(id)    ((id) & 0x0fff0000UL)
#define V4L2_CTRL_DRIVER_PRIV(id) (((id) & 0xffff) >= 0x1000)
#define V4L2_CTRL_MAX_DIMS        (4)
#define V4L2_CTRL_WHICH_CUR_VAL   0
#define V4L2_CTRL_WHICH_DEF_VAL   0x0f000000

enum  v4l2_ctrl_type
 {
         V4L2_CTRL_TYPE_INTEGER
       = 1,
         V4L2_CTRL_TYPE_BOOLEAN
       = 2,
         V4L2_CTRL_TYPE_MENU
          = 3,
         V4L2_CTRL_TYPE_BUTTON
        = 4,
         V4L2_CTRL_TYPE_INTEGER64
     = 5,
         V4L2_CTRL_TYPE_CTRL_CLASS
    = 6,
         V4L2_CTRL_TYPE_STRING
        = 7,
         V4L2_CTRL_TYPE_BITMASK
       = 8,
         V4L2_CTRL_TYPE_INTEGER_MENU
```

```
  = 9,

        /* Compound types are >= 0x0100 */
         V4L2_CTRL_COMPOUND_TYPES      = 0x0100,
         V4L2_CTRL_TYPE_U8
            = 0x0100,
         V4L2_CTRL_TYPE_U16
           = 0x0101,
         V4L2_CTRL_TYPE_U32
           = 0x0102,
};

/*  Used in the  VIDIOC_QUERYCTRL  ioctl for querying controls */
struct  v4l2_queryctrl
 {
        __u32                id;
        __u32                type;      /* enum  v4l2_ctrl_type
 */
        __u8                 name[32];  /* Whatever */
        __s32                minimum;   /* Note signedness */
        __s32                maximum;
        __s32                step;
        __s32                default_value;
        __u32                flags;
        __u32                reserved[2];
};

/*  Used in the  VIDIOC_QUERY_EXT_CTRL  ioctl for querying extended controls */
struct  v4l2_query_ext_ctrl
 {
        __u32                id;
        __u32                type;
        char                 name[32];
        __s64                minimum;
        __s64                maximum;
        __u64                step;
        __s64                default_value;
        __u32                flags;
        __u32                elem_size;
        __u32                elems;
        __u32                nr_of_dims;
        __u32                dims[V4L2_CTRL_MAX_DIMS];
        __u32                reserved[32];
};

/*  Used in the  VIDIOC_QUERYMENU  ioctl for querying menu items */
struct  v4l2_querymenu
 {
        __u32           id;
        __u32           index;
        union {
                __u8    name[32];       /* Whatever */
                __s64   value;
        };
        __u32           reserved;
} __attribute__ ((packed));

/*  Control flags  */
```

```
#define  V4L2_CTRL_FLAG_DISABLED          0x0001
#define  V4L2_CTRL_FLAG_GRABBED           0x0002
#define  V4L2_CTRL_FLAG_READ_ONLY         0x0004
#define  V4L2_CTRL_FLAG_UPDATE            0x0008
#define  V4L2_CTRL_FLAG_INACTIVE          0x0010
#define  V4L2_CTRL_FLAG_SLIDER            0x0020
#define  V4L2_CTRL_FLAG_WRITE_ONLY        0x0040
#define  V4L2_CTRL_FLAG_VOLATILE          0x0080
#define  V4L2_CTRL_FLAG_HAS_PAYLOAD       0x0100
#define  V4L2_CTRL_FLAG_EXECUTE_ON_WRITE  0x0200

/*  Query flags, to be ORed with the control ID */
#define  V4L2_CTRL_FLAG_NEXT_CTRL         0x80000000
#define  V4L2_CTRL_FLAG_NEXT_COMPOUND     0x40000000

/*  User-class control IDs defined by V4L2 */
#define V4L2_CID_MAX_CTRLS              1024
/*  IDs reserved for driver specific controls */
#define  V4L2_CID_PRIVATE_BASE            0x08000000

/*
 *      T U N I N G
 */
struct  v4l2_tuner
 {
        __u32                   index;
        __u8                    name[32];
        __u32                   type;   /* enum  v4l2_tuner_type
 */
        __u32                   capability;
        __u32                   rangelow;
        __u32                   rangehigh;
        __u32                   rxsubchans;
        __u32                   audmode;
        __s32                   signal;
        __s32                   afc;
        __u32                   reserved[4];
};

struct  v4l2_modulator
 {
        __u32                   index;
        __u8                    name[32];
        __u32                   capability;
        __u32                   rangelow;
        __u32                   rangehigh;
        __u32                   txsubchans;
        __u32                   type;   /* enum  v4l2_tuner_type
 */
        __u32                   reserved[3];
};

/*  Flags for the `capability' field */
#define  V4L2_TUNER_CAP_LOW               0x0001
#define  V4L2_TUNER_CAP_NORM              0x0002
#define  V4L2_TUNER_CAP_HWSEEK_BOUNDED    0x0004
#define  V4L2_TUNER_CAP_HWSEEK_WRAP       0x0008
#define  V4L2_TUNER_CAP_STEREO            0x0010
```

```
#define  V4L2_TUNER_CAP_LANG2              0x0020
#define  V4L2_TUNER_CAP_SAP                0x0020
#define  V4L2_TUNER_CAP_LANG1              0x0040
#define  V4L2_TUNER_CAP_RDS                0x0080
#define  V4L2_TUNER_CAP_RDS_BLOCK_IO       0x0100
#define  V4L2_TUNER_CAP_RDS_CONTROLS       0x0200
#define  V4L2_TUNER_CAP_FREQ_BANDS         0x0400
#define  V4L2_TUNER_CAP_HWSEEK_PROG_LIM    0x0800
#define  V4L2_TUNER_CAP_1HZ                0x1000

/*  Flags for the `rxsubchans' field */
#define  V4L2_TUNER_SUB_MONO               0x0001
#define  V4L2_TUNER_SUB_STEREO             0x0002
#define  V4L2_TUNER_SUB_LANG2              0x0004
#define  V4L2_TUNER_SUB_SAP                0x0004
#define  V4L2_TUNER_SUB_LANG1              0x0008
#define  V4L2_TUNER_SUB_RDS                0x0010

/*  Values for the `audmode' field */
#define  V4L2_TUNER_MODE_MONO              0x0000
#define  V4L2_TUNER_MODE_STEREO            0x0001
#define  V4L2_TUNER_MODE_LANG2             0x0002
#define  V4L2_TUNER_MODE_SAP               0x0002
#define  V4L2_TUNER_MODE_LANG1             0x0003
#define  V4L2_TUNER_MODE_LANG1_LANG2       0x0004

struct  v4l2_frequency
 {
        __u32   tuner;
        __u32   type;   /* enum  v4l2_tuner_type
 */
        __u32   frequency;
        __u32   reserved[8];
};

#define  V4L2_BAND_MODULATION_VSB          (1 << 1)
#define  V4L2_BAND_MODULATION_FM           (1 << 2)
#define  V4L2_BAND_MODULATION_AM           (1 << 3)

struct  v4l2_frequency_band
 {
        __u32   tuner;
        __u32   type;   /* enum  v4l2_tuner_type
 */
        __u32   index;
        __u32   capability;
        __u32   rangelow;
        __u32   rangehigh;
        __u32   modulation;
        __u32   reserved[9];
};

struct  v4l2_hw_freq_seek
 {
        __u32   tuner;
        __u32   type;   /* enum  v4l2_tuner_type
 */
        __u32   seek_upward;
```

```
        __u32   wrap_around;
        __u32   spacing;
        __u32   rangelow;
        __u32   rangehigh;
        __u32   reserved[5];
};

/*
 *      R D S
 */

struct  v4l2_rds_data
 {
        __u8    lsb;
        __u8    msb;
        __u8    block;
} __attribute__ ((packed));

#define  V4L2_RDS_BLOCK_MSK        0x7
#define  V4L2_RDS_BLOCK_A          0
#define  V4L2_RDS_BLOCK_B          1
#define  V4L2_RDS_BLOCK_C          2
#define  V4L2_RDS_BLOCK_D          3
#define  V4L2_RDS_BLOCK_C_ALT      4
#define  V4L2_RDS_BLOCK_INVALID    7

#define  V4L2_RDS_BLOCK_CORRECTED  0x40
#define  V4L2_RDS_BLOCK_ERROR      0x80

/*
 *      A U D I O
 */
struct  v4l2_audio
 {
        __u32   index;
        __u8    name[32];
        __u32   capability;
        __u32   mode;
        __u32   reserved[2];
};

/*  Flags for the `capability' field */
#define  V4L2_AUDCAP_STEREO              0x00001
#define  V4L2_AUDCAP_AVL                 0x00002

/*  Flags for the `mode' field */
#define  V4L2_AUDMODE_AVL                0x00001

struct  v4l2_audioout
 {
        __u32   index;
        __u8    name[32];
        __u32   capability;
        __u32   mode;
        __u32   reserved[2];
};

/*
```

```
 *      M P E G   S E R V I C E S
 */
#if 1
#define  V4L2_ENC_IDX_FRAME_I
    (0)
#define  V4L2_ENC_IDX_FRAME_P
    (1)
#define  V4L2_ENC_IDX_FRAME_B
    (2)
#define  V4L2_ENC_IDX_FRAME_MASK
 (0xf)

struct  v4l2_enc_idx_entry
 {
        __u64 offset;
        __u64 pts;
        __u32 length;
        __u32 flags;
        __u32 reserved[2];
};

#define  V4L2_ENC_IDX_ENTRIES
 (64)
struct  v4l2_enc_idx
 {
        __u32 entries;
        __u32 entries_cap;
        __u32 reserved[4];
        struct  v4l2_enc_idx_entry
 entry[V4L2_ENC_IDX_ENTRIES];
};

#define  V4L2_ENC_CMD_START       (0)
#define  V4L2_ENC_CMD_STOP        (1)
#define  V4L2_ENC_CMD_PAUSE       (2)
#define  V4L2_ENC_CMD_RESUME      (3)

/* Flags for  V4L2_ENC_CMD_STOP  */
#define  V4L2_ENC_CMD_STOP_AT_GOP_END     (1 << 0)

struct  v4l2_encoder_cmd
 {
        __u32 cmd;
        __u32 flags;
        union {
                struct {
                        __u32 data[8];
                } raw;
        };
};

/* Decoder commands */
#define  V4L2_DEC_CMD_START        (0)
#define  V4L2_DEC_CMD_STOP         (1)
#define  V4L2_DEC_CMD_PAUSE        (2)
#define  V4L2_DEC_CMD_RESUME       (3)

/* Flags for  V4L2_DEC_CMD_START  */
```

```
#define  V4L2_DEC_CMD_START_MUTE_AUDIO    (1 << 0)

/* Flags for  V4L2_DEC_CMD_PAUSE  */
#define  V4L2_DEC_CMD_PAUSE_TO_BLACK      (1 << 0)

/* Flags for  V4L2_DEC_CMD_STOP  */
#define  V4L2_DEC_CMD_STOP_TO_BLACK       (1 << 0)
#define  V4L2_DEC_CMD_STOP_IMMEDIATELY    (1 << 1)

/* Play format requirements (returned by the driver): */

/* The decoder has no special format requirements */
#define  V4L2_DEC_START_FMT_NONE          (0)
/* The decoder requires full GOPs */
#define  V4L2_DEC_START_FMT_GOP           (1)

/* The structure must be zeroed before use by the application
   This ensures it can be extended safely in the future. */
struct  v4l2_decoder_cmd
 {
        __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 nor-
mal speed. */
                        __s32 speed;
                        __u32 format;
                } start;

                struct {
                        __u32 data[16];
                } raw;
        };
};
#endif

/*
 *      D A T A   S E R V I C E S   ( V B I )
 *
 *      Data services API by Michael Schimek
 */

/* Raw VBI */
struct  v4l2_vbi_format
 {
        __u32   sampling_rate;          /* in 1 Hz */
        __u32   offset;
        __u32   samples_per_line;
        __u32   sample_format;          /* V4L2_PIX_FMT_* */
```

```
        __s32   start[2];
        __u32   count[2];
        __u32   flags;                  /* V4L2_VBI_* */
        __u32   reserved[2];            /* must be zero */
};

/*  VBI flags  */
#define  V4L2_VBI_UNSYNC          (1 << 0)
#define  V4L2_VBI_INTERLACED      (1 << 1)

/* ITU-R start lines for each field */
#define  V4L2_VBI_ITU_525_F1_START
 (1)
#define  V4L2_VBI_ITU_525_F2_START
 (264)
#define  V4L2_VBI_ITU_625_F1_START
 (1)
#define  V4L2_VBI_ITU_625_F2_START
 (314)

/* Sliced VBI
 *
 *    This implements is a proposal V4L2 API to allow SLICED VBI
 * required for some hardware encoders. It should change without
 * notice in the definitive implementation.
 */

struct  v4l2_sliced_vbi_format
 {
        __u16   service_set;
        /* service_lines[0][...] specifies lines 0-23 (1-23 used) of the first field
           service_lines[1][...] specifies lines 0-23 (1-23 used) of the second field
                                 (equals frame lines 313-336 for 625 line video
                                  standards, 263-286 for 525 line standards) */
        __u16   service_lines[2][24];
        __u32   io_size;
        __u32   reserved[2];            /* must be zero */
};

/* Teletext World System Teletext
   (WST), defined on ITU-R BT.653-2 */
#define  V4L2_SLICED_TELETEXT_B           (0x0001)
/* Video Program System, defined on ETS 300 231*/
#define  V4L2_SLICED_VPS                  (0x0400)
/* Closed Caption, defined on EIA-608 */
#define  V4L2_SLICED_CAPTION_525          (0x1000)
/* Wide Screen System, defined on ITU-R BT1119.1 */
#define  V4L2_SLICED_WSS_625              (0x4000)

#define  V4L2_SLICED_VBI_525              ( V4L2_SLICED_CAPTION_525 )
#define  V4L2_SLICED_VBI_625              ( V4L2_SLICED_TELETEXT_B  |  V4L2_SLICED_VPS
|  V4L2_SLICED_WSS_625 )

struct  v4l2_sliced_vbi_cap
 {
        __u16   service_set;
        /* service_lines[0][...] specifies lines 0-23 (1-23 used) of the first field
           service_lines[1][...] specifies lines 0-23 (1-23 used) of the second field
```

```
                                (equals frame lines 313-336 for 625 line video
                                 standards, 263-286 for 525 line standards) */
        __u16   service_lines[2][24];
        __u32   type;           /* enum  v4l2_buf_type
 */
        __u32   reserved[3];    /* must be 0 */
};

struct  v4l2_sliced_vbi_data
 {
        __u32   id;
        __u32   field;          /* 0: first field, 1: second field */
        __u32   line;           /* 1-23 */
        __u32   reserved;       /* must be 0 */
        __u8    data[48];
};

/*
 * Sliced VBI data inserted into MPEG Streams
 */

/*
 * V4L2_MPEG_STREAM_VBI_FMT_IVTV:
 *
 * Structure of payload contained in an MPEG 2 Private Stream 1 PES Packet in an
 * MPEG-2 Program Pack that contains V4L2_MPEG_STREAM_VBI_FMT_IVTV Sliced VBI
 * data
 *
 * Note, the MPEG-2 Program Pack and Private Stream 1 PES packet header
 * definitions are not included here.  See the MPEG-2 specifications for details
 * on these headers.
 */

/* Line type IDs */
#define  V4L2_MPEG_VBI_IVTV_TELETEXT_B      (1)
#define  V4L2_MPEG_VBI_IVTV_CAPTION_525     (4)
#define  V4L2_MPEG_VBI_IVTV_WSS_625         (5)
#define  V4L2_MPEG_VBI_IVTV_VPS             (7)

struct  v4l2_mpeg_vbi_itv0_line
 {
        __u8 id;        /* One of V4L2_MPEG_VBI_IVTV_* above */
        __u8 data[42];  /* Sliced VBI data for the line */
} __attribute__ ((packed));

struct  v4l2_mpeg_vbi_itv0
 {
        __le32 linemask[2]; /* Bitmasks of VBI service lines present */
        struct  v4l2_mpeg_vbi_itv0_line
 line[35];
} __attribute__ ((packed));

struct  v4l2_mpeg_vbi_ITV0
 {
        struct  v4l2_mpeg_vbi_itv0_line
 line[36];
} __attribute__ ((packed));
```

```
#define  V4L2_MPEG_VBI_IVTV_MAGIC0        ``itv0''
#define  V4L2_MPEG_VBI_IVTV_MAGIC1        ``ITV0''

struct  v4l2_mpeg_vbi_fmt_ivtv
 {
        __u8 magic[4];
        union {
                struct  v4l2_mpeg_vbi_itv0
 itv0;
                struct  v4l2_mpeg_vbi_ITV0
 ITV0;
        };
} __attribute__ ((packed));

/*
 *      A G G R E G A T E   S T R U C T U R E S
 */

/**
 * struct  v4l2_plane_pix_format
 - additional, per-plane format definition
 * @sizeimage:          maximum size in bytes required for data, for which
 *                      this plane will be used
 * @bytesperline:       distance in bytes between the leftmost pixels in two
 *                      adjacent lines
 */
struct  v4l2_plane_pix_format
 {
        __u32           sizeimage;
        __u32           bytesperline;
        __u16           reserved[6];
} __attribute__ ((packed));

/**
 * struct  v4l2_pix_format_mplane
 - multiplanar format definition
 * @width:              image width in pixels
 * @height:             image height in pixels
 * @pixelformat:        little endian four character code (fourcc)
 * @field:              enum  v4l2_field
; field order (for interlaced video)
 * @colorspace:         enum  v4l2_colorspace
; supplemental to pixelformat
 * @plane_fmt:          per-plane information
 * @num_planes:         number of planes for this format
 * @flags:              format flags (V4L2_PIX_FMT_FLAG_*)
 * @ycbcr_enc:          enum  v4l2_ycbcr_encoding
, Y'CbCr encoding
 * @quantization:       enum  v4l2_quantization
, colorspace quantization
 * @xfer_func:          enum  v4l2_xfer_func
, colorspace transfer function
 */
struct  v4l2_pix_format_mplane
 {
        __u32                           width;
        __u32                           height;
        __u32                           pixelformat;
```

```
        __u32                           field;
        __u32                           colorspace;

        struct  v4l2_plane_pix_format
    plane_fmt[VIDEO_MAX_PLANES];
        __u8                            num_planes;
        __u8                            flags;
         union {
                __u8                            ycbcr_enc;
                __u8                            hsv_enc;
        };
        __u8                            quantization;
        __u8                            xfer_func;
        __u8                            reserved[7];
} __attribute__ ((packed));

/**
 * struct  v4l2_sdr_format
 - SDR format definition
 * @pixelformat:        little endian four character code (fourcc)
 * @buffersize:         maximum size in bytes required for data
 */
struct  v4l2_sdr_format
 {
        __u32                           pixelformat;
        __u32                           buffersize;
        __u8                            reserved[24];
} __attribute__ ((packed));

/**
 * struct  v4l2_format
 - stream data format
 * @type:       enum  v4l2_buf_type
; type of the data stream
 * @pix:        definition of an image format
 * @pix_mp:     definition of a multiplanar image format
 * @win:        definition of an overlaid image
 * @vbi:        raw VBI capture or output parameters
 * @sliced:     sliced VBI capture or output parameters
 * @raw_data:   placeholder for future extensions and custom formats
 */
struct  v4l2_format
 {
        __u32    type;
        union {
                struct  v4l2_pix_format
          pix;     /*  V4L2_BUF_TYPE_VIDEO_CAPTURE
 */
                struct  v4l2_pix_format_mplane
   pix_mp;  /*  V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE
 */
                struct  v4l2_window
              win;     /*  V4L2_BUF_TYPE_VIDEO_OVERLAY
 */
                struct  v4l2_vbi_format
          vbi;     /*  V4L2_BUF_TYPE_VBI_CAPTURE
 */
                struct  v4l2_sliced_vbi_format
```

```
   sliced;  /*  V4L2_BUF_TYPE_SLICED_VBI_CAPTURE
 */
                struct  v4l2_sdr_format
          sdr;     /*  V4L2_BUF_TYPE_SDR_CAPTURE
 */
                __u8    raw_data[200];                   /* user-defined */
        } fmt;
};

/*      Stream type-dependent parameters
 */
struct  v4l2_streamparm
 {
        __u32    type;                  /* enum  v4l2_buf_type
 */
        union {
                struct  v4l2_captureparm
 capture;
                struct  v4l2_outputparm
  output;
                __u8    raw_data[200];  /* user-defined */
        } parm;
};

/*
 *      E V E N T S
 */

#define  V4L2_EVENT_ALL                         0
#define  V4L2_EVENT_VSYNC                       1
#define  V4L2_EVENT_EOS                         2
#define  V4L2_EVENT_CTRL                        3
#define  V4L2_EVENT_FRAME_SYNC                  4
#define  V4L2_EVENT_SOURCE_CHANGE               5
#define  V4L2_EVENT_MOTION_DET                  6
#define  V4L2_EVENT_PRIVATE_START               0x08000000

/* Payload for  V4L2_EVENT_VSYNC  */
struct  v4l2_event_vsync
 {
        /* Can be  V4L2_FIELD_ANY
, _NONE, _TOP or _BOTTOM */
        __u8 field;
} __attribute__ ((packed));

/* Payload for  V4L2_EVENT_CTRL  */
#define  V4L2_EVENT_CTRL_CH_VALUE               (1 << 0)
#define  V4L2_EVENT_CTRL_CH_FLAGS               (1 << 1)
#define  V4L2_EVENT_CTRL_CH_RANGE               (1 << 2)

struct  v4l2_event_ctrl
 {
        __u32 changes;
        __u32 type;
        union {
                __s32 value;
                __s64 value64;
        };
```

```
        __u32 flags;
        __s32 minimum;
        __s32 maximum;
        __s32 step;
        __s32 default_value;
};

struct  v4l2_event_frame_sync
 {
        __u32 frame_sequence;
};

#define  V4L2_EVENT_SRC_CH_RESOLUTION             (1 << 0)

struct  v4l2_event_src_change
 {
        __u32 changes;
};

#define  V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ
 (1 << 0)

/**
 * struct  v4l2_event_motion_det
 - motion detection event
 * @flags:             if  V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ
 is set, then the
 *                     frame_sequence field is valid.
 * @frame_sequence:    the frame sequence number associated with this event.
 * @region_mask:       which regions detected motion.
 */
struct  v4l2_event_motion_det
 {
        __u32 flags;
        __u32 frame_sequence;
        __u32 region_mask;
};

struct  v4l2_event
 {
        __u32                           type;
        union {
                struct  v4l2_event_vsync
         vsync;
                struct  v4l2_event_ctrl
          ctrl;
                struct  v4l2_event_frame_sync
    frame_sync;
                struct  v4l2_event_src_change
    src_change;
                struct  v4l2_event_motion_det
    motion_det;
                __u8                            data[64];
        } u;
        __u32                           pending;
        __u32                           sequence;
        struct timespec                 timestamp;
        __u32                           id;
```

```
        __u32                           reserved[8];
};

#define  V4L2_EVENT_SUB_FL_SEND_INITIAL           (1 << 0)
#define  V4L2_EVENT_SUB_FL_ALLOW_FEEDBACK         (1 << 1)

struct  v4l2_event_subscription
 {
        __u32                           type;
        __u32                           id;
        __u32                           flags;
        __u32                           reserved[5];
};

/*
 *      A D V A N C E D   D E B U G G I N G
 *
 *      NOTE: EXPERIMENTAL API, NEVER RELY ON THIS IN APPLICATIONS!
 *      FOR DEBUGGING, TESTING AND INTERNAL USE ONLY!
 */

/*  VIDIOC_DBG_G_REGISTER  and  VIDIOC_DBG_S_REGISTER  */

#define  V4L2_CHIP_MATCH_BRIDGE       0  /* Match against chip ID on the bridge (0 for the br
#define  V4L2_CHIP_MATCH_SUBDEV       4  /* Match against subdev index */

/* The following four defines are no longer in use */
#define  V4L2_CHIP_MATCH_HOST   V4L2_CHIP_MATCH_BRIDGE
#define  V4L2_CHIP_MATCH_I2C_DRIVER   1  /* Match against I2C driver name */
#define  V4L2_CHIP_MATCH_I2C_ADDR     2  /* Match against I2C 7-bit address */
#define  V4L2_CHIP_MATCH_AC97         3  /* Match against ancillary AC97 chip */

struct  v4l2_dbg_match
 {
        __u32 type; /* Match type */
        union {     /* Match this chip, meaning determined by type */
                __u32 addr;
                char name[32];
        };
} __attribute__ ((packed));

struct  v4l2_dbg_register
 {
        struct  v4l2_dbg_match
 match;
        __u32 size;     /* register size in bytes */
        __u64 reg;
        __u64 val;
} __attribute__ ((packed));

#define  V4L2_CHIP_FL_READABLE  (1 << 0)
#define  V4L2_CHIP_FL_WRITABLE  (1 << 1)

/*  VIDIOC_DBG_G_CHIP_INFO  */
struct  v4l2_dbg_chip_info
 {
        struct  v4l2_dbg_match
 match;
```

```
        char name[32];
        __u32 flags;
        __u32 reserved[32];
} __attribute__ ((packed));

/**
 * struct  v4l2_create_buffers
 -  VIDIOC_CREATE_BUFS  argument
 * @index:      on return, index of the first created buffer
 * @count:      entry: number of requested buffers,
 *              return: number of created buffers
 * @memory:     enum  v4l2_memory
; buffer memory type
 * @format:     frame format, for which buffers are requested
 * @reserved:   future extensions
 */
struct  v4l2_create_buffers
 {
        __u32                   index;
        __u32                   count;
        __u32                   memory;
        struct  v4l2_format
      format;
        __u32                   reserved[8];
};

/*
 *      I O C T L   C O D E S   F O R   V I D E O   D E V I C E S
 *
 */
#define  VIDIOC_QUERYCAP          _IOR(`V',  0, struct  v4l2_capability
)
#define VIDIOC_RESERVED          _IO(`V',  1)
#define  VIDIOC_ENUM_FMT         _IOWR(`V',  2, struct  v4l2_fmtdesc
)
#define  VIDIOC_G_FMT            _IOWR(`V',  4, struct  v4l2_format
)
#define  VIDIOC_S_FMT            _IOWR(`V',  5, struct  v4l2_format
)
#define  VIDIOC_REQBUFS          _IOWR(`V',  8, struct  v4l2_requestbuffers
)
#define  VIDIOC_QUERYBUF         _IOWR(`V',  9, struct  v4l2_buffer
)
#define  VIDIOC_G_FBUF            _IOR(`V', 10, struct  v4l2_framebuffer
)
#define  VIDIOC_S_FBUF            _IOW(`V', 11, struct  v4l2_framebuffer
)
#define  VIDIOC_OVERLAY           _IOW(`V', 14, int)
#define  VIDIOC_QBUF             _IOWR(`V', 15, struct  v4l2_buffer
)
#define  VIDIOC_EXPBUF           _IOWR(`V', 16, struct  v4l2_exportbuffer
)
#define  VIDIOC_DQBUF            _IOWR(`V', 17, struct  v4l2_buffer
)
#define  VIDIOC_STREAMON          _IOW(`V', 18, int)
#define  VIDIOC_STREAMOFF         _IOW(`V', 19, int)
#define  VIDIOC_G_PARM           _IOWR(`V', 21, struct  v4l2_streamparm
)
```

```
#define  VIDIOC_S_PARM            _IOWR('V', 22, struct  v4l2_streamparm
)
#define  VIDIOC_G_STD              _IOR('V', 23, v4l2_std_id)
#define  VIDIOC_S_STD              _IOW('V', 24, v4l2_std_id)
#define  VIDIOC_ENUMSTD           _IOWR('V', 25, struct  v4l2_standard
)
#define  VIDIOC_ENUMINPUT         _IOWR('V', 26, struct  v4l2_input
)
#define  VIDIOC_G_CTRL            _IOWR('V', 27, struct  v4l2_control
)
#define  VIDIOC_S_CTRL            _IOWR('V', 28, struct  v4l2_control
)
#define  VIDIOC_G_TUNER           _IOWR('V', 29, struct  v4l2_tuner
)
#define  VIDIOC_S_TUNER            _IOW('V', 30, struct  v4l2_tuner
)
#define  VIDIOC_G_AUDIO            _IOR('V', 33, struct  v4l2_audio
)
#define  VIDIOC_S_AUDIO            _IOW('V', 34, struct  v4l2_audio
)
#define  VIDIOC_QUERYCTRL         _IOWR('V', 36, struct  v4l2_queryctrl
)
#define  VIDIOC_QUERYMENU         _IOWR('V', 37, struct  v4l2_querymenu
)
#define  VIDIOC_G_INPUT            _IOR('V', 38, int)
#define  VIDIOC_S_INPUT           _IOWR('V', 39, int)
#define  VIDIOC_G_EDID            _IOWR('V', 40, struct v4l2_edid)
#define  VIDIOC_S_EDID            _IOWR('V', 41, struct v4l2_edid)
#define  VIDIOC_G_OUTPUT           _IOR('V', 46, int)
#define  VIDIOC_S_OUTPUT          _IOWR('V', 47, int)
#define  VIDIOC_ENUMOUTPUT        _IOWR('V', 48, struct  v4l2_output
)
#define  VIDIOC_G_AUDOUT           _IOR('V', 49, struct  v4l2_audioout
)
#define  VIDIOC_S_AUDOUT           _IOW('V', 50, struct  v4l2_audioout
)
#define  VIDIOC_G_MODULATOR       _IOWR('V', 54, struct  v4l2_modulator
)
#define  VIDIOC_S_MODULATOR        _IOW('V', 55, struct  v4l2_modulator
)
#define  VIDIOC_G_FREQUENCY       _IOWR('V', 56, struct  v4l2_frequency
)
#define  VIDIOC_S_FREQUENCY        _IOW('V', 57, struct  v4l2_frequency
)
#define  VIDIOC_CROPCAP           _IOWR('V', 58, struct  v4l2_cropcap
)
#define  VIDIOC_G_CROP            _IOWR('V', 59, struct  v4l2_crop
)
#define  VIDIOC_S_CROP             _IOW('V', 60, struct  v4l2_crop
)
#define  VIDIOC_G_JPEGCOMP         _IOR('V', 61, struct  v4l2_jpegcompression
)
#define  VIDIOC_S_JPEGCOMP         _IOW('V', 62, struct  v4l2_jpegcompression
)
#define  VIDIOC_QUERYSTD           _IOR('V', 63, v4l2_std_id)
#define  VIDIOC_TRY_FMT           _IOWR('V', 64, struct  v4l2_format
)
#define  VIDIOC_ENUMAUDIO         _IOWR('V', 65, struct  v4l2_audio
```

```
)
#define  VIDIOC_ENUMAUDOUT        _IOWR(`V', 66, struct  v4l2_audioout
)
#define  VIDIOC_G_PRIORITY         _IOR(`V', 67, __u32) /* enum  v4l2_priority
 */
#define  VIDIOC_S_PRIORITY         _IOW(`V', 68, __u32) /* enum  v4l2_priority
 */
#define  VIDIOC_G_SLICED_VBI_CAP  _IOWR(`V', 69, struct  v4l2_sliced_vbi_cap
)
#define  VIDIOC_LOG_STATUS          _IO(`V', 70)
#define  VIDIOC_G_EXT_CTRLS       _IOWR(`V', 71, struct  v4l2_ext_controls
)
#define  VIDIOC_S_EXT_CTRLS       _IOWR(`V', 72, struct  v4l2_ext_controls
)
#define  VIDIOC_TRY_EXT_CTRLS     _IOWR(`V', 73, struct  v4l2_ext_controls
)
#define  VIDIOC_ENUM_FRAMESIZES   _IOWR(`V', 74, struct  v4l2_frmsizeenum
)
#define  VIDIOC_ENUM_FRAMEINTERVALS  _IOWR(`V', 75, struct  v4l2_frmivalenum
)
#define  VIDIOC_G_ENC_INDEX        _IOR(`V', 76, struct  v4l2_enc_idx
)
#define  VIDIOC_ENCODER_CMD       _IOWR(`V', 77, struct  v4l2_encoder_cmd
)
#define  VIDIOC_TRY_ENCODER_CMD   _IOWR(`V', 78, struct  v4l2_encoder_cmd
)

/*
 * Experimental, meant for debugging, testing and internal use.
 * Only implemented if CONFIG_VIDEO_ADV_DEBUG is defined.
 * You must be root to use these ioctls. Never use these in applications!
 */
#define  VIDIOC_DBG_S_REGISTER     _IOW(`V', 79, struct  v4l2_dbg_register
)
#define  VIDIOC_DBG_G_REGISTER    _IOWR(`V', 80, struct  v4l2_dbg_register
)

#define  VIDIOC_S_HW_FREQ_SEEK     _IOW(`V', 82, struct  v4l2_hw_freq_seek
)
#define  VIDIOC_S_DV_TIMINGS      _IOWR(`V', 87, struct  v4l2_dv_timings
)
#define  VIDIOC_G_DV_TIMINGS      _IOWR(`V', 88, struct  v4l2_dv_timings
)
#define  VIDIOC_DQEVENT            _IOR(`V', 89, struct  v4l2_event
)
#define  VIDIOC_SUBSCRIBE_EVENT    _IOW(`V', 90, struct  v4l2_event_subscription
)
#define  VIDIOC_UNSUBSCRIBE_EVENT  _IOW(`V', 91, struct  v4l2_event_subscription
)
#define  VIDIOC_CREATE_BUFS       _IOWR(`V', 92, struct  v4l2_create_buffers
)
#define  VIDIOC_PREPARE_BUF       _IOWR(`V', 93, struct  v4l2_buffer
)
#define  VIDIOC_G_SELECTION       _IOWR(`V', 94, struct  v4l2_selection
)
#define  VIDIOC_S_SELECTION       _IOWR(`V', 95, struct  v4l2_selection
)
#define  VIDIOC_DECODER_CMD       _IOWR(`V', 96, struct  v4l2_decoder_cmd
```

```
)
#define  VIDIOC_TRY_DECODER_CMD   _IOWR(`V', 97, struct  v4l2_decoder_cmd
)
#define  VIDIOC_ENUM_DV_TIMINGS   _IOWR(`V', 98, struct  v4l2_enum_dv_timings
)
#define  VIDIOC_QUERY_DV_TIMINGS   _IOR(`V', 99, struct  v4l2_dv_timings
)
#define  VIDIOC_DV_TIMINGS_CAP    _IOWR(`V', 100, struct  v4l2_dv_timings_cap
)
#define  VIDIOC_ENUM_FREQ_BANDS   _IOWR(`V', 101, struct  v4l2_frequency_band
)

/*
 * Experimental, meant for debugging, testing and internal use.
 * Never use this in applications!
 */
#define  VIDIOC_DBG_G_CHIP_INFO   _IOWR(`V', 102, struct  v4l2_dbg_chip_info
)

#define  VIDIOC_QUERY_EXT_CTRL    _IOWR(`V', 103, struct  v4l2_query_ext_ctrl
)

/* Reminder: when adding new ioctls please add support for them to
   drivers/media/v4l2-core/v4l2-compat-ioctl32.c as well! */

#define BASE_VIDIOC_PRIVATE     192             /* 192-255 are private */

#endif /* _UAPI__LINUX_VIDEODEV2_H */
```

1.2.11 Video Capture Example

file: media/v4l/capture.c

```
/*
 *  V4L2 video capture example
 *
 *  This program can be used and distributed without restrictions.
 *
 *      This program is provided with the V4L2 API
 * see https://linuxtv.org/docs.php for more information
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include <getopt.h>             /* getopt_long() */

#include <fcntl.h>              /* low-level i/o */
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

#include <linux/videodev2.h>
```

```
#define CLEAR(x) memset(&(x), 0, sizeof(x))

enum io_method {
        IO_METHOD_READ,
        IO_METHOD_MMAP,
        IO_METHOD_USERPTR,
};

struct buffer {
        void   *start;
        size_t  length;
};

static char            *dev_name;
static enum io_method   io = IO_METHOD_MMAP;
static int              fd = -1;
struct buffer          *buffers;
static unsigned int     n_buffers;
static int              out_buf;
static int              force_format;
static int              frame_count = 70;

static void errno_exit(const char *s)
{
        fprintf(stderr, "%s error %d, %s\\n", s, errno, strerror(errno));
        exit(EXIT_FAILURE);
}

static int xioctl(int fh, int request, void *arg)
{
        int r;

        do {
                r = ioctl(fh, request, arg);
        } while (-1 == r && EINTR == errno);

        return r;
}

static void process_image(const void *p, int size)
{
        if (out_buf)
                fwrite(p, size, 1, stdout);

        fflush(stderr);
        fprintf(stderr, ".");
        fflush(stdout);
}

static int read_frame(void)
{
        struct v4l2_buffer buf;
        unsigned int i;

        switch (io) {
        case IO_METHOD_READ:
                if (-1 == read(fd, buffers[0].start, buffers[0].length)) {
                        switch (errno) {
                        case EAGAIN:
                                return 0;

                        case EIO:
```

```
                        /* Could ignore EIO, see spec. */

                        /* fall through */

                default:
                        errno_exit("read");
                }
        }

        process_image(buffers[0].start, buffers[0].length);
        break;

    case IO_METHOD_MMAP:
        CLEAR(buf);

        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory = V4L2_MEMORY_MMAP;

        if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
                switch (errno) {
                case EAGAIN:
                        return 0;

                case EIO:
                        /* Could ignore EIO, see spec. */

                        /* fall through */

                default:
                        errno_exit("VIDIOC_DQBUF");
                }
        }

        assert(buf.index < n_buffers);

        process_image(buffers[buf.index].start, buf.bytesused);

        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                errno_exit("VIDIOC_QBUF");
        break;

    case IO_METHOD_USERPTR:
        CLEAR(buf);

        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory = V4L2_MEMORY_USERPTR;

        if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
                switch (errno) {
                case EAGAIN:
                        return 0;

                case EIO:
                        /* Could ignore EIO, see spec. */

                        /* fall through */

                default:
                        errno_exit("VIDIOC_DQBUF");
                }
        }

        for (i = 0; i < n_buffers; ++i)
```

```
                        if (buf.m.userptr == (unsigned long)buffers[i].start
                            && buf.length == buffers[i].length)
                                break;

                assert(i < n_buffers);

                process_image((void *)buf.m.userptr, buf.bytesused);

                if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                        errno_exit("VIDIOC_QBUF");
                break;
        }

        return 1;
}

static void mainloop(void)
{
        unsigned int count;

        count = frame_count;

        while (count-- > 0) {
                for (;;) {
                        fd_set fds;
                        struct timeval tv;
                        int r;

                        FD_ZERO(&fds);
                        FD_SET(fd, &fds);

                        /* Timeout. */
                        tv.tv_sec = 2;
                        tv.tv_usec = 0;

                        r = select(fd + 1, &fds, NULL, NULL, &tv);

                        if (-1 == r) {
                                if (EINTR == errno)
                                        continue;
                                errno_exit("select");
                        }

                        if (0 == r) {
                                fprintf(stderr, "select timeout\\n");
                                exit(EXIT_FAILURE);
                        }

                        if (read_frame())
                                break;
                        /* EAGAIN - continue select loop. */
                }
        }
}

static void stop_capturing(void)
{
        enum v4l2_buf_type type;

        switch (io) {
        case IO_METHOD_READ:
                /* Nothing to do. */
                break;
```

```
        case IO_METHOD_MMAP:
        case IO_METHOD_USERPTR:
                type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                if (-1 == xioctl(fd, VIDIOC_STREAMOFF, &type))
                        errno_exit("VIDIOC_STREAMOFF");
                break;
        }
}

static void start_capturing(void)
{
        unsigned int i;
        enum v4l2_buf_type type;

        switch (io) {
        case IO_METHOD_READ:
                /* Nothing to do. */
                break;

        case IO_METHOD_MMAP:
                for (i = 0; i < n_buffers; ++i) {
                        struct v4l2_buffer buf;

                        CLEAR(buf);
                        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                        buf.memory = V4L2_MEMORY_MMAP;
                        buf.index = i;

                        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                                errno_exit("VIDIOC_QBUF");
                }
                type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
                        errno_exit("VIDIOC_STREAMON");
                break;

        case IO_METHOD_USERPTR:
                for (i = 0; i < n_buffers; ++i) {
                        struct v4l2_buffer buf;

                        CLEAR(buf);
                        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                        buf.memory = V4L2_MEMORY_USERPTR;
                        buf.index = i;
                        buf.m.userptr = (unsigned long)buffers[i].start;
                        buf.length = buffers[i].length;

                        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                                errno_exit("VIDIOC_QBUF");
                }
                type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
                        errno_exit("VIDIOC_STREAMON");
                break;
        }
}

static void uninit_device(void)
{
        unsigned int i;

        switch (io) {
```

```
        case IO_METHOD_READ:
                free(buffers[0].start);
                break;

        case IO_METHOD_MMAP:
                for (i = 0; i < n_buffers; ++i)
                        if (-1 == munmap(buffers[i].start, buffers[i].length))
                                errno_exit("munmap");
                break;

        case IO_METHOD_USERPTR:
                for (i = 0; i < n_buffers; ++i)
                        free(buffers[i].start);
                break;
        }

        free(buffers);
}

static void init_read(unsigned int buffer_size)
{
        buffers = calloc(1, sizeof(*buffers));

        if (!buffers) {
                fprintf(stderr, "Out of memory\\n");
                exit(EXIT_FAILURE);
        }

        buffers[0].length = buffer_size;
        buffers[0].start = malloc(buffer_size);

        if (!buffers[0].start) {
                fprintf(stderr, "Out of memory\\n");
                exit(EXIT_FAILURE);
        }
}

static void init_mmap(void)
{
        struct v4l2_requestbuffers req;

        CLEAR(req);

        req.count = 4;
        req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        req.memory = V4L2_MEMORY_MMAP;

        if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
                if (EINVAL == errno) {
                        fprintf(stderr, "%s does not support "
                                 "memory mappingn", dev_name);
                        exit(EXIT_FAILURE);
                } else {
                        errno_exit("VIDIOC_REQBUFS");
                }
        }

        if (req.count < 2) {
                fprintf(stderr, "Insufficient buffer memory on %s\\n",
                         dev_name);
                exit(EXIT_FAILURE);
        }
```

```
        buffers = calloc(req.count, sizeof(*buffers));

        if (!buffers) {
                fprintf(stderr, "Out of memory\\n");
                exit(EXIT_FAILURE);
        }

        for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
                struct v4l2_buffer buf;

                CLEAR(buf);

                buf.type        = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory      = V4L2_MEMORY_MMAP;
                buf.index       = n_buffers;

                if (-1 == xioctl(fd, VIDIOC_QUERYBUF, &buf))
                        errno_exit("VIDIOC_QUERYBUF");

                buffers[n_buffers].length = buf.length;
                buffers[n_buffers].start =
                        mmap(NULL /* start anywhere */,
                              buf.length,
                              PROT_READ | PROT_WRITE /* required */,
                              MAP_SHARED /* recommended */,
                              fd, buf.m.offset);

                if (MAP_FAILED == buffers[n_buffers].start)
                        errno_exit("mmap");
        }
}

static void init_userp(unsigned int buffer_size)
{
        struct v4l2_requestbuffers req;

        CLEAR(req);

        req.count  = 4;
        req.type   = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        req.memory = V4L2_MEMORY_USERPTR;

        if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
                if (EINVAL == errno) {
                        fprintf(stderr, "%s does not support "
                                 "user pointer i/on", dev_name);
                        exit(EXIT_FAILURE);
                } else {
                        errno_exit("VIDIOC_REQBUFS");
                }
        }

        buffers = calloc(4, sizeof(*buffers));

        if (!buffers) {
                fprintf(stderr, "Out of memory\\n");
                exit(EXIT_FAILURE);
        }

        for (n_buffers = 0; n_buffers < 4; ++n_buffers) {
                buffers[n_buffers].length = buffer_size;
                buffers[n_buffers].start = malloc(buffer_size);
```

```
                if (!buffers[n_buffers].start) {
                        fprintf(stderr, "Out of memory\\n");
                        exit(EXIT_FAILURE);
                }
        }
}

static void init_device(void)
{
        struct v4l2_capability cap;
        struct v4l2_cropcap cropcap;
        struct v4l2_crop crop;
        struct v4l2_format fmt;
        unsigned int min;

        if (-1 == xioctl(fd, VIDIOC_QUERYCAP, &cap)) {
                if (EINVAL == errno) {
                        fprintf(stderr, "%s is no V4L2 device\\n",
                                dev_name);
                        exit(EXIT_FAILURE);
                } else {
                        errno_exit("VIDIOC_QUERYCAP");
                }
        }

        if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
                fprintf(stderr, "%s is no video capture device\\n",
                         dev_name);
                exit(EXIT_FAILURE);
        }

        switch (io) {
        case IO_METHOD_READ:
                if (!(cap.capabilities & V4L2_CAP_READWRITE)) {
                        fprintf(stderr, "%s does not support read i/o\\n",
                                dev_name);
                        exit(EXIT_FAILURE);
                }
                break;

        case IO_METHOD_MMAP:
        case IO_METHOD_USERPTR:
                if (!(cap.capabilities & V4L2_CAP_STREAMING)) {
                        fprintf(stderr, "%s does not support streaming i/o\\n",
                                dev_name);
                        exit(EXIT_FAILURE);
                }
                break;
        }


        /* Select video input, video standard and tune here. */


        CLEAR(cropcap);

        cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

        if (0 == xioctl(fd, VIDIOC_CROPCAP, &cropcap)) {
                crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                crop.c = cropcap.defrect; /* reset to default */

                if (-1 == xioctl(fd, VIDIOC_S_CROP, &crop)) {
```

```
                        switch (errno) {
                        case EINVAL:
                                /* Cropping not supported. */
                                break;
                        default:
                                /* Errors ignored. */
                                break;
                        }
                }
        } else {
                /* Errors ignored. */
        }


        CLEAR(fmt);

        fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (force_format) {
                fmt.fmt.pix.width       = 640;
                fmt.fmt.pix.height      = 480;
                fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
                fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;

                if (-1 == xioctl(fd, VIDIOC_S_FMT, &fmt))
                        errno_exit("VIDIOC_S_FMT");

                /* Note VIDIOC_S_FMT may change width and height. */
        } else {
                /* Preserve original settings as set by v4l2-ctl for example */
                if (-1 == xioctl(fd, VIDIOC_G_FMT, &fmt))
                        errno_exit("VIDIOC_G_FMT");
        }

        /* Buggy driver paranoia. */
        min = fmt.fmt.pix.width * 2;
        if (fmt.fmt.pix.bytesperline < min)
                fmt.fmt.pix.bytesperline = min;
        min = fmt.fmt.pix.bytesperline * fmt.fmt.pix.height;
        if (fmt.fmt.pix.sizeimage < min)
                fmt.fmt.pix.sizeimage = min;

        switch (io) {
        case IO_METHOD_READ:
                init_read(fmt.fmt.pix.sizeimage);
                break;

        case IO_METHOD_MMAP:
                init_mmap();
                break;

        case IO_METHOD_USERPTR:
                init_userp(fmt.fmt.pix.sizeimage);
                break;
        }
}

static void close_device(void)
{
        if (-1 == close(fd))
                errno_exit("close");

        fd = -1;
}
```

```
static void open_device(void)
{
        struct stat st;

        if (-1 == stat(dev_name, &st)) {
                fprintf(stderr, "Cannot identify '%s': %d, %s\\n",
                        dev_name, errno, strerror(errno));
                exit(EXIT_FAILURE);
        }

        if (!S_ISCHR(st.st_mode)) {
                fprintf(stderr, "%s is no devicen", dev_name);
                exit(EXIT_FAILURE);
        }

        fd = open(dev_name, O_RDWR /* required */ | O_NONBLOCK, 0);

        if (-1 == fd) {
                fprintf(stderr, "Cannot open '%s': %d, %s\\n",
                        dev_name, errno, strerror(errno));
                exit(EXIT_FAILURE);
        }
}

static void usage(FILE *fp, int argc, char **argv)
{
        fprintf(fp,
                 "Usage: %s [options]\\n\\n"
                 "Version 1.3\\n"
                 "Options:\\n"
                 "-d | --device name   Video device name [%s]n"
                 "-h | --help          Print this messagen"
                 "-m | --mmap          Use memory mapped buffers [default]n"
                 "-r | --read          Use read() callsn"
                 "-u | --userp         Use application allocated buffersn"
                 "-o | --output        Outputs stream to stdoutn"
                 "-f | --format        Force format to 640x480 YUYVn"
                 "-c | --count         Number of frames to grab [%i]n"
                 "",
                 argv[0], dev_name, frame_count);
}

static const char short_options[] = "d:hmruofc:";

static const struct option
long_options[] = {
        { "device", required_argument, NULL, 'd' },
        { "help",   no_argument,       NULL, 'h' },
        { "mmap",   no_argument,       NULL, 'm' },
        { "read",   no_argument,       NULL, 'r' },
        { "userp",  no_argument,       NULL, 'u' },
        { "output", no_argument,       NULL, 'o' },
        { "format", no_argument,       NULL, 'f' },
        { "count",  required_argument, NULL, 'c' },
        { 0, 0, 0, 0 }
};

int main(int argc, char **argv)
{
        dev_name = "/dev/video0";

        for (;;) {
```

```
                int idx;
                int c;

                c = getopt_long(argc, argv,
                                short_options, long_options, &idx);

                if (-1 == c)
                        break;

                switch (c) {
                case 0: /* getopt_long() flag */
                        break;

                case 'd':
                        dev_name = optarg;
                        break;

                case 'h':
                        usage(stdout, argc, argv);
                        exit(EXIT_SUCCESS);

                case 'm':
                        io = IO_METHOD_MMAP;
                        break;

                case 'r':
                        io = IO_METHOD_READ;
                        break;

                case 'u':
                        io = IO_METHOD_USERPTR;
                        break;

                case 'o':
                        out_buf++;
                        break;

                case 'f':
                        force_format++;
                        break;

                case 'c':
                        errno = 0;
                        frame_count = strtol(optarg, NULL, 0);
                        if (errno)
                                errno_exit(optarg);
                        break;

                default:
                        usage(stderr, argc, argv);
                        exit(EXIT_FAILURE);
                }
        }

        open_device();
        init_device();
        start_capturing();
        mainloop();
        stop_capturing();
        uninit_device();
        close_device();
        fprintf(stderr, "\\n");
        return 0;
```

```
}
```

1.2.12 Video Grabber example using libv4l

This program demonstrates how to grab V4L2 images in ppm format by using libv4l handlers. The advantage is that this grabber can potentially work with any V4L2 driver.

file: media/v4l/v4l2grab.c

```
/* V4L2 video picture grabber
   Copyright (C) 2009 Mauro Carvalho Chehab <mchehab@infradead.org>

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation version 2 of the License.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <linux/videodev2.h>
#include "../libv4l/include/libv4l2.h"

#define CLEAR(x) memset(&(x), 0, sizeof(x))

struct buffer {
        void   *start;
        size_t length;
};

static void xioctl(int fh, int request, void *arg)
{
        int r;

        do {
                r = v4l2_ioctl(fh, request, arg);
        } while (r == -1 && ((errno == EINTR) || (errno == EAGAIN)));

        if (r == -1) {
                fprintf(stderr, "error %d, %s\\n", errno, strerror(errno));
                exit(EXIT_FAILURE);
        }
}

int main(int argc, char **argv)
{
        struct v4l2_format              fmt;
        struct v4l2_buffer              buf;
```

```
        struct v4l2_requestbuffers      req;
        enum v4l2_buf_type              type;
        fd_set                          fds;
        struct timeval                  tv;
        int                             r, fd = -1;
        unsigned int                    i, n_buffers;
        char                            *dev_name = "/dev/video0";
        char                            out_name[256];
        FILE                            *fout;
        struct buffer                   *buffers;

        fd = v4l2_open(dev_name, O_RDWR | O_NONBLOCK, 0);
        if (fd < 0) {
                perror("Cannot open device");
                exit(EXIT_FAILURE);
        }

        CLEAR(fmt);
        fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        fmt.fmt.pix.width       = 640;
        fmt.fmt.pix.height      = 480;
        fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB24;
        fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;
        xioctl(fd, VIDIOC_S_FMT, &fmt);
        if (fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_RGB24) {
                printf("Libv4l didn't accept RGB24 format. Can't proceed.\\n");
                exit(EXIT_FAILURE);
        }
        if ((fmt.fmt.pix.width != 640) || (fmt.fmt.pix.height != 480))
                printf("Warning: driver is sending image at %dx%d\\n",
                        fmt.fmt.pix.width, fmt.fmt.pix.height);

        CLEAR(req);
        req.count = 2;
        req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        req.memory = V4L2_MEMORY_MMAP;
        xioctl(fd, VIDIOC_REQBUFS, &req);

        buffers = calloc(req.count, sizeof(*buffers));
        for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
                CLEAR(buf);

                buf.type        = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory      = V4L2_MEMORY_MMAP;
                buf.index       = n_buffers;

                xioctl(fd, VIDIOC_QUERYBUF, &buf);

                buffers[n_buffers].length = buf.length;
                buffers[n_buffers].start = v4l2_mmap(NULL, buf.length,
                              PROT_READ | PROT_WRITE, MAP_SHARED,
                              fd, buf.m.offset);

                if (MAP_FAILED == buffers[n_buffers].start) {
                        perror("mmap");
                        exit(EXIT_FAILURE);
                }
        }

        for (i = 0; i < n_buffers; ++i) {
                CLEAR(buf);
                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory = V4L2_MEMORY_MMAP;
```

```
                buf.index = i;
                xioctl(fd, VIDIOC_QBUF, &buf);
        }
        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

        xioctl(fd, VIDIOC_STREAMON, &type);
        for (i = 0; i < 20; i++) {
                do {
                        FD_ZERO(&fds);
                        FD_SET(fd, &fds);

                        /* Timeout. */
                        tv.tv_sec = 2;
                        tv.tv_usec = 0;

                        r = select(fd + 1, &fds, NULL, NULL, &tv);
                } while ((r == -1 && (errno = EINTR)));
                if (r == -1) {
                        perror("select");
                        return errno;
                }

                CLEAR(buf);
                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory = V4L2_MEMORY_MMAP;
                xioctl(fd, VIDIOC_DQBUF, &buf);

                sprintf(out_name, "out%03d.ppm", i);
                fout = fopen(out_name, "w");
                if (!fout) {
                        perror("Cannot open image");
                        exit(EXIT_FAILURE);
                }
                fprintf(fout, "P6\\n%d %d 255\\n",
                        fmt.fmt.pix.width, fmt.fmt.pix.height);
                fwrite(buffers[buf.index].start, buf.bytesused, 1, fout);
                fclose(fout);

                xioctl(fd, VIDIOC_QBUF, &buf);
        }

        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        xioctl(fd, VIDIOC_STREAMOFF, &type);
        for (i = 0; i < n_buffers; ++i)
                v4l2_munmap(buffers[i].start, buffers[i].length);
        v4l2_close(fd);

        return 0;
}
```

1.2.13 References

CEA 608-E

title CEA-608-E R-2014 “Line 21 Data Services”

author Consumer Electronics Association (http://www.ce.org)

EN 300 294

title EN 300 294 “625-line television Wide Screen Signalling (WSS)”

author European Telecommunication Standards Institute (http://www.etsi.org)

ETS 300 231

title ETS 300 231 “Specification of the domestic video Programme Delivery Control system (PDC)”

author European Telecommunication Standards Institute (http://www.etsi.org)

ETS 300 706

title ETS 300 706 “Enhanced Teletext specification”

author European Telecommunication Standards Institute (http://www.etsi.org)

ISO 13818-1

title ITU-T Rec. H.222.0 | ISO/IEC 13818-1 “Information technology — Generic coding of moving pictures and associated audio information: Systems”

author International Telecommunication Union (http://www.itu.ch), International Organisation for Standardisation (http://www.iso.ch)

ISO 13818-2

title ITU-T Rec. H.262 | ISO/IEC 13818-2 “Information technology — Generic coding of moving pictures and associated audio information: Video”

author International Telecommunication Union (http://www.itu.ch), International Organisation for Standardisation (http://www.iso.ch)

ITU BT.470

title ITU-R Recommendation BT.470-6 “Conventional Television Systems”

author International Telecommunication Union (http://www.itu.ch)

ITU BT.601

title ITU-R Recommendation BT.601-5 “Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide-Screen 16:9 Aspect Ratios”

author International Telecommunication Union (http://www.itu.ch)

ITU BT.653

title ITU-R Recommendation BT.653-3 “Teletext systems”

author International Telecommunication Union (http://www.itu.ch)

ITU BT.709

title ITU-R Recommendation BT.709-5 “Parameter values for the HDTV standards for production and international programme exchange”

author International Telecommunication Union (http://www.itu.ch)

ITU BT.1119

title ITU-R Recommendation BT.1119 “625-line television Wide Screen Signalling (WSS)”

author International Telecommunication Union (http://www.itu.ch)

JFIF

title JPEG File Interchange Format

subtitle Version 1.02

author Independent JPEG Group (http://www.ijg.org)

ITU-T.81

title ITU-T Recommendation T.81 “Information Technology — Digital Compression and Coding of Continous-Tone Still Images — Requirements and Guidelines”

author International Telecommunication Union (http://www.itu.int)

W3C JPEG JFIF

title JPEG JFIF

author The World Wide Web Consortium (http://www.w3.org)

SMPTE 12M

title SMPTE 12M-1999 “Television, Audio and Film - Time and Control Code”

author Society of Motion Picture and Television Engineers (http://www.smpte.org)

SMPTE 170M

title SMPTE 170M-1999 “Television - Composite Analog Video Signal - NTSC for Studio Applications”

author Society of Motion Picture and Television Engineers (http://www.smpte.org)

SMPTE 240M

title SMPTE 240M-1999 “Television - Signal Parameters - 1125-Line High-Definition Production”

author Society of Motion Picture and Television Engineers (http://www.smpte.org)

SMPTE RP 431-2

title SMPTE RP 431-2:2011 “D-Cinema Quality - Reference Projector and Environment”

author Society of Motion Picture and Television Engineers (http://www.smpte.org)

SMPTE ST 2084

title SMPTE ST 2084:2014 “High Dynamic Range Electro-Optical Transfer Function of Master Reference Displays”

author Society of Motion Picture and Television Engineers (http://www.smpte.org)

sRGB

title IEC 61966-2-1 ed1.0 “Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB”

author International Electrotechnical Commission (http://www.iec.ch)

sYCC

title IEC 61966-2-1-am1 ed1.0 “Amendment 1 - Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB”

author International Electrotechnical Commission (http://www.iec.ch)

xvYCC

title IEC 61966-2-4 ed1.0 “Multimedia systems and equipment - Colour measurement and management - Part 2-4: Colour management - Extended-gamut YCC colour space for video applications - xvYCC”

author International Electrotechnical Commission (http://www.iec.ch)

AdobeRGB

title Adobe© RGB (1998) Color Image Encoding Version 2005-05

author Adobe Systems Incorporated (http://www.adobe.com)

opRGB

title IEC 61966-2-5 “Multimedia systems and equipment - Colour measurement and management - Part 2-5: Colour management - Optional RGB colour space - opRGB”

author International Electrotechnical Commission (http://www.iec.ch)

ITU BT.2020

title ITU-R Recommendation BT.2020 (08/2012) “Parameter values for ultra-high definition television systems for production and international programme exchange”

author International Telecommunication Union (http://www.itu.ch)

EBU Tech 3213

title E.B.U. Standard for Chromaticity Tolerances for Studio Monitors”

author European Broadcast Union (http://www.ebu.ch)

IEC 62106

title Specification of the radio data system (RDS) for VHF/FM sound broadcasting in the frequency range from 87,5 to 108,0 MHz

author International Electrotechnical Commission (http://www.iec.ch)

NRSC-4-B

title NRSC-4-B: United States RBDS Standard

author National Radio Systems Committee (http://www.nrscstandards.org)

ISO 12232:2006

title Photography — Digital still cameras — Determination of exposure index, ISO speed ratings, standard output sensitivity, and recommended exposure index

author International Organization for Standardization (http://www.iso.org)

CEA-861-E

title A DTV Profile for Uncompressed High Speed Digital Interfaces

author Consumer Electronics Association (http://www.ce.org)

VESA DMT

title VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT)

author Video Electronics Standards Association (http://www.vesa.org)

EDID

title VESA Enhanced Extended Display Identification Data Standard

subtitle Release A, Revision 2

author Video Electronics Standards Association (http://www.vesa.org)

HDCP

title High-bandwidth Digital Content Protection System

subtitle Revision 1.3

author Digital Content Protection LLC (http://www.digital-cp.com)

HDMI

title High-Definition Multimedia Interface

subtitle Specification Version 1.4a

author HDMI Licensing LLC (http://www.hdmi.org)

HDMI2

title High-Definition Multimedia Interface

subtitle Specification Version 2.0

author HDMI Licensing LLC (http://www.hdmi.org)

DP

title VESA DisplayPort Standard

subtitle Version 1, Revision 2

author Video Electronics Standards Association (http://www.vesa.org)

poynton

title Digital Video and HDTV, Algorithms and Interfaces

author Charles Poynton

colimg

title Color Imaging: Fundamentals and Applications

author Erik Reinhard et al.

1.2.14 Revision and Copyright

Authors, in alphabetical order:

- Ailus, Sakari <sakari.ailus@iki.fi>
  - Subdev selections API.
- Carvalho Chehab, Mauro <m.chehab@kernel.org>
  - Documented libv4l, designed and added v4l2grab example, Remote Controller chapter.
- Dirks, Bill
  - Original author of the V4L2 API and documentation.
- H Schimek, Michael <mschimek@gmx.at>
  - Original author of the V4L2 API and documentation.
- Karicheri, Muralidharan <m-karicheri2@ti.com>
  - Documented the Digital Video timings API.
- Osciak, Pawel <pawel@osciak.com>
  - Designed and documented the multi-planar API.
- Palosaari, Antti <crope@iki.fi>
  - SDR API.
- Ribalda, Ricardo
  - Introduce HSV formats and other minor changes.
- Rubli, Martin

  - Designed and documented the VIDIOC_ENUM_FRAMESIZES and VIDIOC_ENUM_FRAMEINTERVALS ioctls.

- Walls, Andy <awalls@md.metrocast.net>
  - Documented the fielded V4L2_MPEG_STREAM_VBI_FMT_IVTV MPEG stream embedded, sliced VBI data format in this specification.
- Verkuil, Hans <hverkuil@xs4all.nl>
  - Designed and documented the VIDIOC_LOG_STATUS ioctl, the extended control ioctls, major parts of the sliced VBI API, the MPEG encoder and decoder APIs and the DV Timings API.

1.2.15 Revision History

revision 4.10 / 2016-07-15 (*rr*)

Introduce HSV formats.

revision 4.5 / 2015-10-29 (*rr*)

Extend VIDIOC_G_EXT_CTRLS;. Replace ctrl_class with a new union with ctrl_class and which. Which is used to select the current value of the control or the default value.

revision 4.4 / 2015-05-26 (*ap*)

Renamed V4L2_TUNER_ADC to V4L2_TUNER_SDR. Added V4L2_CID_RF_TUNER_RF_GAIN control. Added transmitter support for Software Defined Radio (SDR) Interface.

revision 4.1 / 2015-02-13 (*mcc*)

Fix documentation for media controller device nodes and add support for DVB device nodes. Add support for Tuner sub-device.

revision 3.19 / 2014-12-05 (*hv*)

Rewrote Colorspace chapter, added new enum *v4l2_ycbcr_encoding* and enum *v4l2_quantization* fields to struct *v4l2_pix_format*, struct *v4l2_pix_format_mplane* and struct *v4l2_mbus_framefmt*.

revision 3.17 / 2014-08-04 (*lp, hv*)

Extended struct *v4l2_pix_format*. Added format flags. Added compound control types and VIDIOC_QUERY_EXT_CTRL.

revision 3.15 / 2014-02-03 (*hv, ap*)

Update several sections of “Common API Elements”: “Opening and Closing Devices” “Querying Capabilities”, “Application Priority”, “Video Inputs and Outputs”, “Audio Inputs and Outputs” “Tuners and Modulators”, “Video Standards” and “Digital Video (DV) Timings”. Added SDR API.

revision 3.14 / 2013-11-25 (*rr*)

Set width and height as unsigned on v4l2_rect.

revision 3.11 / 2013-05-26 (*hv*)

Remove obsolete VIDIOC_DBG_G_CHIP_IDENT ioctl.

revision 3.10 / 2013-03-25 (*hv*)

Remove obsolete and unused DV_PRESET ioctls: VIDIOC_G_DV_PRESET, VIDIOC_S_DV_PRESET, VIDIOC_QUERY_DV_PRESET and VIDIOC_ENUM_DV_PRESET. Remove the related v4l2_input/output capability flags V4L2_IN_CAP_PRESETS and V4L2_OUT_CAP_PRESETS. Added VIDIOC_DBG_G_CHIP_INFO.

revision 3.9 / 2012-12-03 (*sa, sn*)

Added timestamp types to v4l2_buffer. Added V4L2_EVENT_CTRL_CH_RANGE control event changes flag.

revision 3.6 / 2012-07-02 (*hv*)

Added VIDIOC_ENUM_FREQ_BANDS.

revision 3.5 / 2012-05-07 (*sa, sn, hv*)

Added V4L2_CTRL_TYPE_INTEGER_MENU and V4L2 subdev selections API. Improved the description of V4L2_CID_COLORFX control, added V4L2_CID_COLORFX_CBCR control. Added camera controls V4L2_CID_AUTO_EXPOSURE_BIAS, V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE, V4L2_CID_IMAGE_STABILIZATION, V4L2_CID_ISO_SENSITIVITY, V4L2_CID_ISO_SENSITIVITY_AUTO, V4L2_CID_EXPOSURE_METERING, V4L2_CID_SCENE_MODE, V4L2_CID_3A_LOCK, V4L2_CID_AUTO_FOCUS_START, V4L2_CID_AUTO_FOCUS_STOP, V4L2_CID_AUTO_FOCUS_STATUS and V4L2_CID_AUTO_FOCUS_RANGE. Added VIDIOC_ENUM_DV_TIMINGS, VIDIOC_QUERY_DV_TIMINGS and VIDIOC_DV_TIMINGS_CAP.

revision 3.4 / 2012-01-25 (*sn*)

Added *JPEG compression control class.*

revision 3.3 / 2012-01-11 (*hv*)

Added device_caps field to struct v4l2_capabilities.

revision 3.2 / 2011-08-26 (*hv*)

Added V4L2_CTRL_FLAG_VOLATILE.

revision 3.1 / 2011-06-27 (*mcc, po, hv*)

Documented that VIDIOC_QUERYCAP now returns a per-subsystem version instead of a per-driver one. Standardize an error code for invalid ioctl. Added V4L2_CTRL_TYPE_BITMASK.

revision 2.6.39 / 2011-03-01 (*mcc, po*)

Removed VIDIOC_*_OLD from videodev2.h header and update it to reflect latest changes. Added the *multi-planar API* .

revision 2.6.37 / 2010-08-06 (*hv*)

Removed obsolete vtx (videotext) API.

revision 2.6.33 / 2009-12-03 (*mk*)

Added documentation for the Digital Video timings API.

revision 2.6.32 / 2009-08-31 (*mcc*)

Now, revisions will match the kernel version where the V4L2 API changes will be used by the Linux Kernel. Also added Remote Controller chapter.

revision 0.29 / 2009-08-26 (*ev*)

Added documentation for string controls and for FM Transmitter controls.

revision 0.28 / 2009-08-26 (*gl*)

Added V4L2_CID_BAND_STOP_FILTER documentation.

revision 0.27 / 2009-08-15 (*mcc*)

Added libv4l and Remote Controller documentation; added v4l2grab and keytable application examples.

revision 0.26 / 2009-07-23 (*hv*)

Finalized the RDS capture API. Added modulator and RDS encoder capabilities. Added support for string controls.

revision 0.25 / 2009-01-18 (*hv*)

Added pixel formats VYUY, NV16 and NV61, and changed the debug ioctls VIDIOC_DBG_G/S_REGISTER and VIDIOC_DBG_G_CHIP_IDENT. Added camera controls V4L2_CID_ZOOM_ABSOLUTE, V4L2_CID_ZOOM_RELATIVE, V4L2_CID_ZOOM_CONTINUOUS and V4L2_CID_PRIVACY.

revision 0.24 / 2008-03-04 (*mhs*)

Added pixel formats Y16 and SBGGR16, new controls and a camera controls class. Removed VIDIOC_G/S_MPEGCOMP.

revision 0.23 / 2007-08-30 (*mhs*)

Fixed a typo in VIDIOC_DBG_G/S_REGISTER. Clarified the byte order of packed pixel formats.

revision 0.22 / 2007-08-29 (*mhs*)

Added the Video Output Overlay interface, new MPEG controls, V4L2_FIELD_INTERLACED_TB and V4L2_FIELD_INTERLACED_BT, VIDIOC_DBG_G/S_REGISTER, VIDIOC_(TRY_)ENCODER_CMD, VIDIOC_G_CHIP_IDENT, VIDIOC_G_ENC_INDEX, new pixel formats. Clarifications in the cropping chapter, about RGB pixel formats, the mmap(), poll(), select(), read() and write() functions. Typographical fixes.

revision 0.21 / 2006-12-19 (*mhs*)

Fixed a link in the VIDIOC_G_EXT_CTRLS section.

revision 0.20 / 2006-11-24 (*mhs*)

Clarified the purpose of the audioset field in struct v4l2_input and v4l2_output.

revision 0.19 / 2006-10-19 (*mhs*)

Documented V4L2_PIX_FMT_RGB444.

revision 0.18 / 2006-10-18 (*mhs*)

Added the description of extended controls by Hans Verkuil. Linked V4L2_PIX_FMT_MPEG to V4L2_CID_MPEG_STREAM_TYPE.

revision 0.17 / 2006-10-12 (*mhs*)

Corrected V4L2_PIX_FMT_HM12 description.

revision 0.16 / 2006-10-08 (*mhs*)

VIDIOC_ENUM_FRAMESIZES and VIDIOC_ENUM_FRAMEINTERVALS are now part of the API.

revision 0.15 / 2006-09-23 (*mhs*)

Cleaned up the bibliography, added BT.653 and BT.1119. capture.c/start_capturing() for user pointer I/O did not initialize the buffer index. Documented the V4L MPEG and MJPEG VID_TYPEs and V4L2_PIX_FMT_SBGGR8. Updated the list of reserved pixel formats. See the history chapter for API changes.

revision 0.14 / 2006-09-14 (*mr*)

Added VIDIOC_ENUM_FRAMESIZES and VIDIOC_ENUM_FRAMEINTERVALS proposal for frame format enumeration of digital devices.

revision 0.13 / 2006-04-07 (*mhs*)

Corrected the description of struct v4l2_window clips. New V4L2_STD_ and V4L2_TUNER_MODE_LANG1_LANG2 defines.

revision 0.12 / 2006-02-03 (*mhs*)

Corrected the description of struct v4l2_captureparm and v4l2_outputparm.

revision 0.11 / 2006-01-27 (*mhs*)

Improved the description of struct v4l2_tuner.

revision 0.10 / 2006-01-10 (*mhs*)

VIDIOC_G_INPUT and VIDIOC_S_PARM clarifications.

revision 0.9 / 2005-11-27 (*mhs*)

Improved the 525 line numbering diagram. Hans Verkuil and I rewrote the sliced VBI section. He also contributed a VIDIOC_LOG_STATUS page. Fixed VIDIOC_S_STD call in the video standard selection example. Various updates.

revision 0.8 / 2004-10-04 (*mhs*)

Somehow a piece of junk slipped into the capture example, removed.

revision 0.7 / 2004-09-19 (*mhs*)

Fixed video standard selection, control enumeration, downscaling and aspect example. Added read and user pointer i/o to video capture example.

revision 0.6 / 2004-08-01 (*mhs*)

v4l2_buffer changes, added video capture example, various corrections.

revision 0.5 / 2003-11-05 (*mhs*)

Pixel format erratum.

revision 0.4 / 2003-09-17 (*mhs*)

Corrected source and Makefile to generate a PDF. SGML fixes. Added latest API changes. Closed gaps in the history chapter.

revision 0.3 / 2003-02-05 (*mhs*)

Another draft, more corrections.

revision 0.2 / 2003-01-15 (*mhs*)

Second draft, with corrections pointed out by Gerd Knorr.

revision 0.1 / 2002-12-01 (*mhs*)

First draft, based on documentation by Bill Dirks and discussions on the V4L mailing list.

1.3 Part II - Digital TV API

Note:

*This API is also known as **DVB API**, although it is generic enough to support all digital TV standards.*

Version 5.10

Table of Contents

1.3.1 Introduction

What you need to know

The reader of this document is required to have some knowledge in the area of digital video broadcasting (DVB) and should be familiar with part I of the MPEG2 specification ISO/IEC 13818 (aka ITU-T H.222), i.e you should know what a program/transport stream (PS/TS) is and what is meant by a packetized elementary stream (PES) or an I-frame.

Various DVB standards documents are available from http://www.dvb.org and/or http://www.etsi.org.

It is also necessary to know how to access unix/linux devices and how to use ioctl calls. This also includes the knowledge of C or C++.

History

The first API for DVB cards we used at Convergence in late 1999 was an extension of the Video4Linux API which was primarily developed for frame grabber cards. As such it was not really well suited to be used for DVB cards and their new features like recording MPEG streams and filtering several section and PES data streams at the same time.

In early 2000, we were approached by Nokia with a proposal for a new standard Linux DVB API. As a commitment to the development of terminals based on open standards, Nokia and Convergence made it available to all Linux developers and published it on https://linuxtv.org in September 2000. Convergence is the maintainer of the Linux DVB API. Together with the LinuxTV community (i.e. you, the reader of this document), the Linux DVB API will be constantly reviewed and improved. With the Linux driver for the Siemens/Hauppauge DVB PCI card Convergence provides a first implementation of the Linux DVB API.

Overview

Fig. 1.15: Components of a DVB card/STB

A DVB PCI card or DVB set-top-box (STB) usually consists of the following main hardware components:

- Frontend consisting of tuner and DVB demodulator

  Here the raw signal reaches the DVB hardware from a satellite dish or antenna or directly from cable. The frontend down-converts and demodulates this signal into an MPEG transport stream (TS). In case of a satellite frontend, this includes a facility for satellite equipment control (SEC), which allows control of LNB polarization, multi feed switches or dish rotors.

- Conditional Access (CA) hardware like CI adapters and smartcard slots

  The complete TS is passed through the CA hardware. Programs to which the user has access (controlled by the smart card) are decoded in real time and re-inserted into the TS.

- Demultiplexer which filters the incoming DVB stream

  The demultiplexer splits the TS into its components like audio and video streams. Besides usually several of such audio and video streams it also contains data streams with information about the programs offered in this or other streams of the same provider.

- MPEG2 audio and video decoder

  The main targets of the demultiplexer are the MPEG2 audio and video decoders. After decoding they pass on the uncompressed audio and video to the computer screen or (through a PAL/NTSC encoder) to a TV set.

Components of a DVB card/STB shows a crude schematic of the control and data flow between those components.

On a DVB PCI card not all of these have to be present since some functionality can be provided by the main CPU of the PC (e.g. MPEG picture and sound decoding) or is not needed (e.g. for data-only uses like “internet over satellite”). Also not every card or STB provides conditional access hardware.

Linux DVB Devices

The Linux DVB API lets you control these hardware components through currently six Unix-style character devices for video, audio, frontend, demux, CA and IP-over-DVB networking. The video and audio devices control the MPEG2 decoder hardware, the frontend device the tuner and the DVB demodulator. The demux device gives you control over the PES and section filters of the hardware. If the hardware does not support filtering these filters can be implemented in software. Finally, the CA device controls all the conditional access capabilities of the hardware. It can depend on the individual security requirements of the platform, if and how many of the CA functions are made available to the application through this device.

All devices can be found in the /dev tree under /dev/dvb. The individual devices are called:

- /dev/dvb/adapterN/audioM,
- /dev/dvb/adapterN/videoM,
- /dev/dvb/adapterN/frontendM,
- /dev/dvb/adapterN/netM,
- /dev/dvb/adapterN/demuxM,
- /dev/dvb/adapterN/dvrM,
- /dev/dvb/adapterN/caM,

where N enumerates the DVB PCI cards in a system starting from 0, and M enumerates the devices of each type within each adapter, starting from 0, too. We will omit the " /dev/dvb/adapterN/" in the further discussion of these devices.

More details about the data structures and function calls of all the devices are described in the following chapters.

API include files

For each of the DVB devices a corresponding include file exists. The DVB API include files should be included in application sources with a partial path like:

```
#include <linux/dvb/ca.h>

#include <linux/dvb/dmx.h>

#include <linux/dvb/frontend.h>

#include <linux/dvb/net.h>
```

To enable applications to support different API version, an additional include file linux/dvb/version.h exists, which defines the constant DVB_API_VERSION. This document describes DVB_API_VERSION 5.10.

1.3.2 DVB Frontend API

The DVB frontend API was designed to support three types of delivery systems:

- Terrestrial systems: DVB-T, DVB-T2, ATSC, ATSC M/H, ISDB-T, DVB-H, DTMB, CMMB
- Cable systems: DVB-C Annex A/C, ClearQAM (DVB-C Annex B), ISDB-C
- Satellite systems: DVB-S, DVB-S2, DVB Turbo, ISDB-S, DSS

The DVB frontend controls several sub-devices including:

- Tuner
- Digital TV demodulator
- Low noise amplifier (LNA)
- Satellite Equipment Control (SEC) hardware (only for Satellite).

The frontend can be accessed through /dev/dvb/adapter?/frontend?. Data types and ioctl definitions can be accessed by including linux/dvb/frontend.h in your application.

Note:

Transmission via the internet (DVB-IP) is not yet handled by this API but a future extension is possible.

On Satellite systems, the API support for the Satellite Equipment Control (SEC) allows to power control and to send/receive signals to control the antenna subsystem, selecting the polarization and choosing the Intermediate Frequency IF) of the Low Noise Block Converter Feed Horn (LNBf). It supports the DiSEqC and V-SEC protocols. The DiSEqC (digital SEC) specification is available at Eutelsat.

Querying frontend information

Usually, the first thing to do when the frontend is opened is to check the frontend capabilities. This is done using *ioctl FE_GET_INFO* . This ioctl will enumerate the DVB API version and other characteristics about the frontend, and can be opened either in read only or read/write mode.

Querying frontend status and statistics

Once *FE_SET_PROPERTY* is called, the frontend will run a kernel thread that will periodically check for the tuner lock status and provide statistics about the quality of the signal.

The information about the frontend tuner locking status can be queried using *ioctl FE_READ_STATUS* .

Signal statistics are provided via *ioctl FE_SET_PROPERTY, FE_GET_PROPERTY* .

Note:

Most statistics require the demodulator to be fully locked (e. g. with FE_HAS_LOCK bit set). See Frontend statistics indicators *for more details.*

DVB Frontend properties

Tuning into a Digital TV physical channel and starting decoding it requires changing a set of parameters, in order to control the tuner, the demodulator, the Linear Low-noise Amplifier (LNA) and to set the antenna subsystem via Satellite Equipment Control (SEC), on satellite systems. The actual parameters are specific to each particular digital TV standards, and may change as the digital TV specs evolves.

In the past, the strategy used was to have a union with the parameters needed to tune for DVB-S, DVB-C, DVB-T and ATSC delivery systems grouped there. The problem is that, as the second generation standards appeared, those structs were not big enough to contain the additional parameters. Also, the union didn't have any space left to be expanded without breaking userspace. So, the decision was to deprecate the legacy union/struct based approach, in favor of a properties set approach.

Note:

On Linux DVB API version 3, setting a frontend were done via struct `dvb_frontend_parameters`*. This got replaced on version 5 (also called "S2API", as this API were added originally_enabled to provide support for DVB-S2), because the old API has a very limited support to new standards and new hardware. This section describes the new and recommended way to set the frontend, with suppports all digital TV delivery systems.*

Example: with the properties based approach, in order to set the tuner to a DVB-C channel at 651 kHz, modulated with 256-QAM, FEC 3/4 and symbol rate of 5.217 Mbauds, those properties should be sent to *FE_SET_PROPERTY* ioctl:

- *DTV_DELIVERY_SYSTEM* = SYS_DVBC_ANNEX_A
- *DTV_FREQUENCY* = 651000000
- *DTV_MODULATION* = QAM_256
- *DTV_INVERSION* = INVERSION_AUTO

- *DTV_SYMBOL_RATE* = 5217000
- *DTV_INNER_FEC* = FEC_3_4
- *DTV_TUNE*

The code that would that would do the above is show in *Example: Setting digital TV frontend properties* .

Example: Setting digital TV frontend properties

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/dvb/frontend.h>

static struct dtv_property props[] = {
    { .cmd = DTV_DELIVERY_SYSTEM, .u.data = SYS_DVBC_ANNEX_A },
    { .cmd = DTV_FREQUENCY,       .u.data = 651000000 },
    { .cmd = DTV_MODULATION,      .u.data = QAM_256 },
    { .cmd = DTV_INVERSION,       .u.data = INVERSION_AUTO },
    { .cmd = DTV_SYMBOL_RATE,     .u.data = 5217000 },
    { .cmd = DTV_INNER_FEC,       .u.data = FEC_3_4 },
    { .cmd = DTV_TUNE }
};

static struct dtv_properties dtv_prop = {
    .num = 6, .props = props
};

int main(void)
{
    int fd = open("/dev/dvb/adapter0/frontend0", O_RDWR);

    if (!fd) {
        perror ("open");
        return -1;
    }
    if (ioctl(fd, FE_SET_PROPERTY, &dtv_prop) == -1) {
        perror("ioctl");
        return -1;
    }
    printf("Frontend set\\n");
    return 0;
}
```

Attention:

While it is possible to directly call the Kernel code like the above example, it is strongly recommended to use libdvbv5, as it provides abstraction to work with the supported digital TV standards and provides methods for usual operations like program scanning and to read/write channel descriptor files.

dtv_stats

struct dtv_stats

```
struct dtv_stats {
    __u8 scale; /* enum fecap_scale_params type */
    union {
```

```
        __u64 uvalue;   /* for counters and relative scales */
        __s64 svalue;   /* for 1/1000 dB measures */
    };
} __packed;
```

dtv_fe_stats

struct dtv_fe_stats

```
#define MAX_DTV_STATS   4

struct dtv_fe_stats {
    __u8 len;
    struct dtv_stats stat[MAX_DTV_STATS];
} __packed;
```

dtv_property

struct dtv_property

```
/* Reserved fields should be set to 0 */

struct dtv_property {
    __u32 cmd;
    __u32 reserved[3];
    union {
        __u32 data;
        struct dtv_fe_stats st;
        struct {
            __u8 data[32];
            __u32 len;
            __u32 reserved1[3];
            void *reserved2;
        } buffer;
    } u;
    int result;
} __attribute__ ((packed));

/* num of properties cannot exceed DTV_IOCTL_MAX_MSGS per ioctl */
#define DTV_IOCTL_MAX_MSGS 64
```

dtv_properties

struct dtv_properties

```
struct dtv_properties {
    __u32 num;
    struct dtv_property *props;
};
```

Property types

On *FE_GET_PROPERTY and FE_SET_PROPERTY* , the actual action is determined by the dtv_property cmd/data pairs. With one single ioctl, is possible to get/set up to 64 properties. The actual meaning of each property is described on the next sections.

The available frontend property types are shown on the next section.

Digital TV property parameters

DTV_UNDEFINED

Used internally. A GET/SET operation for it won't change or return anything.

DTV_TUNE

Interpret the cache of data, build either a traditional frontend tunerequest so we can pass validation in the FE_SET_FRONTEND ioctl.

DTV_CLEAR

Reset a cache of data specific to the frontend here. This does not effect hardware.

DTV_FREQUENCY

Frequency of the digital TV transponder/channel.

Note:

1. *For satellite delivery systems, the frequency is in kHz.*
2. *For cable and terrestrial delivery systems, the frequency is in Hz.*
3. *On most delivery systems, the frequency is the center frequency of the transponder/channel. The exception is for ISDB-T, where the main carrier has a 1/7 offset from the center.*
4. *For ISDB-T, the channels are usually transmitted with an offset of about 143kHz. E.g. a valid frequency could be 474,143 kHz. The stepping is bound to the bandwidth of the channel which is typically 6MHz.*
5. *In ISDB-Tsb, the channel consists of only one or three segments the frequency step is 429kHz, 3*429 respectively.*

DTV_MODULATION

Specifies the frontend modulation type for delivery systems that supports more than one modulation type. The modulation can be one of the types defined by enum *fe_modulation*.

fe_modulation

Modulation property

Most of the digital TV standards currently offers more than one possible modulation (sometimes called as "constellation" on some standards). This enum contains the values used by the Kernel. Please note that not all modulations are supported by a given standard.

Table 1.161: enum fe_modulation

ID	Description
QPSK	QPSK modulation
QAM_16	16-QAM modulation
QAM_32	32-QAM modulation
QAM_64	64-QAM modulation
QAM_128	128-QAM modulation
QAM_256	256-QAM modulation
QAM_AUTO	Autodetect QAM modulation
VSB_8	8-VSB modulation
VSB_16	16-VSB modulation
PSK_8	8-PSK modulation
APSK_16	16-APSK modulation
APSK_32	32-APSK modulation
DQPSK	DQPSK modulation
QAM_4_NR	4-QAM-NR modulation

DTV_BANDWIDTH_HZ

Bandwidth for the channel, in HZ.

Possible values: 1712000, 5000000, 6000000, 7000000, 8000000, 10000000.

Note:

1. *DVB-T supports 6, 7 and 8MHz.*
2. *DVB-T2 supports 1.172, 5, 6, 7, 8 and 10MHz.*
3. *ISDB-T supports 5MHz, 6MHz, 7MHz and 8MHz, although most places use 6MHz.*
4. *On DVB-C and DVB-S/S2, the bandwidth depends on the symbol rate. So, the Kernel will silently ignore setting* DTV_BANDWIDTH_HZ *.*
5. *For DVB-C and DVB-S/S2, the Kernel will return an estimation of the bandwidth, calculated from* DTV_SYMBOL_RATE *and from the rolloff, with is fixed for DVB-C and DVB-S.*
6. *For DVB-S2, the bandwidth estimation will use* DTV_ROLLOFF *.*
7. *For ISDB-Tsb, it can vary depending on the number of connected segments.*
8. *Bandwidth in ISDB-Tsb can be easily derived from other parameters (DTV_ISDBT_SB_SEGMENT_IDX, DTV_ISDBT_SB_SEGMENT_COUNT).*

DTV_INVERSION

Specifies if the frontend should do spectral inversion or not.

fe_spectral_inversion

enum fe_modulation: Frontend spectral inversion

This parameter indicates if spectral inversion should be presumed or not. In the automatic setting (INVERSION_AUTO) the hardware will try to figure out the correct setting by itself. If the hardware doesn't support, the DVB core will try to lock at the carrier first with inversion off. If it fails, it will try to enable inversion.

Table 1.162: enum fe_modulation

ID	Description
INVERSION_OFF	Don't do spectral band inversion.
INVERSION_ON	Do spectral band inversion.
INVERSION_AUTO	Autodetect spectral band inversion.

DTV_DISEQC_MASTER

Currently not implemented.

DTV_SYMBOL_RATE

Digital TV symbol rate, in bauds (symbols/second). Used on cable standards.

DTV_INNER_FEC

Used cable/satellite transmissions. The acceptable values are:

fe_code_rate

enum fe_code_rate: type of the Forward Error Correction.

Table 1.163: enum fe_code_rate

ID	Description
FEC_NONE	No Forward Error Correction Code
FEC_AUTO	Autodetect Error Correction Code
FEC_1_2	Forward Error Correction Code 1/2
FEC_2_3	Forward Error Correction Code 2/3
FEC_3_4	Forward Error Correction Code 3/4
FEC_4_5	Forward Error Correction Code 4/5
FEC_5_6	Forward Error Correction Code 5/6
FEC_6_7	Forward Error Correction Code 6/7
FEC_7_8	Forward Error Correction Code 7/8
FEC_8_9	Forward Error Correction Code 8/9
FEC_9_10	Forward Error Correction Code 9/10
FEC_2_5	Forward Error Correction Code 2/5
FEC_3_5	Forward Error Correction Code 3/5

DTV_VOLTAGE

The voltage is usually used with non-DiSEqC capable LNBs to switch the polarzation (horizontal/vertical). When using DiSEqC epuipment this voltage has to be switched consistently to the DiSEqC commands as described in the DiSEqC spec.

fe_sec_voltage

Table 1.164: enum fe_sec_voltage

ID	Description
SEC_VOLTAGE_13	Set DC voltage level to 13V
SEC_VOLTAGE_18	Set DC voltage level to 18V
SEC_VOLTAGE_OFF	Don’t send any voltage to the antenna

DTV_TONE

Currently not used.

DTV_PILOT

Sets DVB-S2 pilot

fe_pilot

fe_pilot type

Table 1.165: enum fe_pilot

ID	Description
PILOT_ON	Pilot tones enabled
PILOT_OFF	Pilot tones disabled
PILOT_AUTO	Autodetect pilot tones

DTV_ROLLOFF

Sets DVB-S2 rolloff

fe_rolloff

fe_rolloff type

Table 1.166: enum fe_rolloff

ID	Description
ROLLOFF_35	Roloff factor: α=35%
ROLLOFF_20	Roloff factor: α=20%
ROLLOFF_25	Roloff factor: α=25%
ROLLOFF_AUTO	Auto-detect the roloff factor.

DTV_DISEQC_SLAVE_REPLY

Currently not implemented.

DTV_FE_CAPABILITY_COUNT

Currently not implemented.

DTV_FE_CAPABILITY

Currently not implemented.

DTV_DELIVERY_SYSTEM

Specifies the type of Delivery system

fe_delivery_system

fe_delivery_system type

Possible values:

Table 1.167: enum fe_delivery_system

ID	Description
SYS_UNDEFINED	Undefined standard. Generally, indicates an error
SYS_DVBC_ANNEX_A	Cable TV: DVB-C following ITU-T J.83 Annex A spec
SYS_DVBC_ANNEX_B	Cable TV: DVB-C following ITU-T J.83 Annex B spec (ClearQAM)
SYS_DVBC_ANNEX_C	Cable TV: DVB-C following ITU-T J.83 Annex C spec
SYS_ISDBC	Cable TV: ISDB-C (no drivers yet)
SYS_DVBT	Terrestral TV: DVB-T
SYS_DVBT2	Terrestral TV: DVB-T2
SYS_ISDBT	Terrestral TV: ISDB-T
SYS_ATSC	Terrestral TV: ATSC
SYS_ATSCMH	Terrestral TV (mobile): ATSC-M/H
SYS_DTMB	Terrestrial TV: DTMB
SYS_DVBS	Satellite TV: DVB-S
SYS_DVBS2	Satellite TV: DVB-S2
SYS_TURBO	Satellite TV: DVB-S Turbo
SYS_ISDBS	Satellite TV: ISDB-S
SYS_DAB	Digital audio: DAB (not fully supported)
SYS_DSS	Satellite TV:”DSS (not fully supported)
SYS_CMMB	Terrestral TV (mobile):CMMB (not fully supported)
SYS_DVBH	Terrestral TV (mobile): DVB-H (standard deprecated)

DTV_ISDBT_PARTIAL_RECEPTION

If DTV_ISDBT_SOUND_BROADCASTING is ‘0’ this bit-field represents whether the channel is in partial reception mode or not.

If ‘1’ DTV_ISDBT_LAYERA_* values are assigned to the center segment and DTV_ISDBT_LAYERA_SEGMENT_COUNT has to be ‘1’.

If in addition DTV_ISDBT_SOUND_BROADCASTING is ‘1’ DTV_ISDBT_PARTIAL_RECEPTION represents whether this ISDB-Tsb channel is consisting of one segment and layer or three segments and two layers.

Possible values: 0, 1, -1 (AUTO)

DTV_ISDBT_SOUND_BROADCASTING

This field represents whether the other DTV_ISDBT_*-parameters are referring to an ISDB-T and an ISDB-Tsb channel. (See also DTV_ISDBT_PARTIAL_RECEPTION).

Possible values: 0, 1, -1 (AUTO)

DTV_ISDBT_SB_SUBCHANNEL_ID

This field only applies if DTV_ISDBT_SOUND_BROADCASTING is ‘1’.

(Note of the author: This might not be the correct description of the SUBCHANNEL-ID in all details, but it is my understanding of the technical background needed to program a device)

An ISDB-Tsb channel (1 or 3 segments) can be broadcasted alone or in a set of connected ISDB-Tsb channels. In this set of channels every channel can be received independently. The number of connected ISDB-Tsb segment can vary, e.g. depending on the frequency spectrum bandwidth available.

Example: Assume 8 ISDB-Tsb connected segments are broadcasted. The broadcaster has several possibilities to put those channels in the air: Assuming a normal 13-segment ISDB-T spectrum he can align the 8 segments from position 1-8 to 5-13 or anything in between.

The underlying layer of segments are subchannels: each segment is consisting of several subchannels with a predefined IDs. A sub-channel is used to help the demodulator to synchronize on the channel.

An ISDB-T channel is always centered over all sub-channels. As for the example above, in ISDB-Tsb it is no longer as simple as that.

The DTV_ISDBT_SB_SUBCHANNEL_ID parameter is used to give the sub-channel ID of the segment to be demodulated.

Possible values: 0 .. 41, -1 (AUTO)

DTV_ISDBT_SB_SEGMENT_IDX

This field only applies if DTV_ISDBT_SOUND_BROADCASTING is ‘1’.

DTV_ISDBT_SB_SEGMENT_IDX gives the index of the segment to be demodulated for an ISDB-Tsb channel where several of them are transmitted in the connected manner.

Possible values: 0 .. DTV_ISDBT_SB_SEGMENT_COUNT - 1

Note: This value cannot be determined by an automatic channel search.

DTV_ISDBT_SB_SEGMENT_COUNT

This field only applies if DTV_ISDBT_SOUND_BROADCASTING is ‘1’.

DTV_ISDBT_SB_SEGMENT_COUNT gives the total count of connected ISDB-Tsb channels.

Possible values: 1 .. 13

Note: This value cannot be determined by an automatic channel search.

DTV-ISDBT-LAYER[A-C] parameters

ISDB-T channels can be coded hierarchically. As opposed to DVB-T in ISDB-T hierarchical layers can be decoded simultaneously. For that reason a ISDB-T demodulator has 3 Viterbi and 3 Reed-Solomon decoders.

ISDB-T has 3 hierarchical layers which each can use a part of the available segments. The total number of segments over all layers has to 13 in ISDB-T.

There are 3 parameter sets, for Layers A, B and C.

DTV_ISDBT_LAYER_ENABLED

Hierarchical reception in ISDB-T is achieved by enabling or disabling layers in the decoding process. Setting all bits of DTV_ISDBT_LAYER_ENABLED to '1' forces all layers (if applicable) to be demodulated. This is the default.

If the channel is in the partial reception mode (DTV_ISDBT_PARTIAL_RECEPTION = 1) the central segment can be decoded independently of the other 12 segments. In that mode layer A has to have a SEGMENT_COUNT of 1.

In ISDB-Tsb only layer A is used, it can be 1 or 3 in ISDB-Tsb according to DTV_ISDBT_PARTIAL_RECEPTION. SEGMENT_COUNT must be filled accordingly.

Only the values of the first 3 bits are used. Other bits will be silently ignored:

DTV_ISDBT_LAYER_ENABLED bit 0: layer A enabled

DTV_ISDBT_LAYER_ENABLED bit 1: layer B enabled

DTV_ISDBT_LAYER_ENABLED bit 2: layer C enabled

DTV_ISDBT_LAYER_ENABLED bits 3-31: unused

DTV_ISDBT_LAYER[A-C]_FEC

Possible values: FEC_AUTO, FEC_1_2, FEC_2_3, FEC_3_4, FEC_5_6, FEC_7_8

DTV_ISDBT_LAYER[A-C]_MODULATION

Possible values: QAM_AUTO, QPSK,QAM_16, QAM_64, DQPSK

Note: If layer C is DQPSK layer B has to be DQPSK. If layer B is DQPSK and DTV_ISDBT_PARTIAL_RECEPTION=0 layer has to be DQPSK.

DTV_ISDBT_LAYER[A-C]_SEGMENT_COUNT

Possible values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, -1 (AUTO)

Note: Truth table for DTV_ISDBT_SOUND_BROADCASTING and DTV_ISDBT_PARTIAL_RECEPTION and LAYER[A-C]_SEGMENT_COUNT

Table 1.168: Truth table for ISDB-T Sound Broadcasting

PR	SB	Layer A width	Layer B width	Layer C width	total width
0	0	1 .. 13	1 .. 13	1 .. 13	13
1	0	1	1 .. 13	1 .. 13	13
0	1	1	0	0	1
1	1	1	2	0	13

DTV_ISDBT_LAYER[A-C]_TIME_INTERLEAVING

Valid values: 0, 1, 2, 4, -1 (AUTO)

when DTV_ISDBT_SOUND_BROADCASTING is active, value 8 is also valid.

Note: The real time interleaving length depends on the mode (fft-size). The values here are referring to what can be found in the TMCC-structure, as shown in the table below.

isdbt_layer_interleaving_table

Table 1.169: ISDB-T time interleaving modes

DTV_ISDBT_LAYER[A-C]_TIME_INTERLEAVING	Mode 1 (2K FFT)	Mode 2 (4K FFT)	Mode 3 (8K FFT)
0	0	0	0
1	4	2	1
2	8	4	2
4	16	8	4

DTV_ATSCMH_FIC_VER

Version number of the FIC (Fast Information Channel) signaling data.

FIC is used for relaying information to allow rapid service acquisition by the receiver.

Possible values: 0, 1, 2, 3, ..., 30, 31

DTV_ATSCMH_PARADE_ID

Parade identification number

A parade is a collection of up to eight MH groups, conveying one or two ensembles.

Possible values: 0, 1, 2, 3, ..., 126, 127

DTV_ATSCMH_NOG

Number of MH groups per MH subframe for a designated parade.

Possible values: 1, 2, 3, 4, 5, 6, 7, 8

DTV_ATSCMH_TNOG

Total number of MH groups including all MH groups belonging to all MH parades in one MH subframe.

Possible values: 0, 1, 2, 3, ..., 30, 31

DTV_ATSCMH_SGN

Start group number.

Possible values: 0, 1, 2, 3, ..., 14, 15

DTV_ATSCMH_PRC

Parade repetition cycle.

Possible values: 1, 2, 3, 4, 5, 6, 7, 8

DTV_ATSCMH_RS_FRAME_MODE

Reed Solomon (RS) frame mode.

Possible values are:

atscmh_rs_frame_mode

Table 1.170: enum atscmh_rs_frame_mode

ID	Description
ATSCMH_RSFRAME_PRI_ONLY	Single Frame: There is only a primary RS Frame for all Group Regions.
ATSCMH_RSFRAME_PRI_SEC	Dual Frame: There are two separate RS Frames: Primary RS Frame for Group Region A and B and Secondary RS Frame for Group Region C and D.

DTV_ATSCMH_RS_FRAME_ENSEMBLE

Reed Solomon(RS) frame ensemble.

Possible values are:

atscmh_rs_frame_ensemble

Table 1.171: enum atscmh_rs_frame_ensemble

ID	Description
ATSCMH_RSFRAME_ENS_PRI	Primary Ensemble.
AATSCMH_RSFRAME_PRI_SEC	Secondary Ensemble.
AATSCMH_RSFRAME_RES	Reserved. Shouldn't be used.

DTV_ATSCMH_RS_CODE_MODE_PRI

Reed Solomon (RS) code mode (primary).

Possible values are:

atscmh_rs_code_mode

Table 1.172: enum atscmh_rs_code_mode

ID	Description
ATSCMH_RSCODE_211_187	Reed Solomon code (211,187).
ATSCMH_RSCODE_223_187	Reed Solomon code (223,187).
ATSCMH_RSCODE_235_187	Reed Solomon code (235,187).
ATSCMH_RSCODE_RES	Reserved. Shouldn't be used.

DTV_ATSCMH_RS_CODE_MODE_SEC

Reed Solomon (RS) code mode (secondary).

Possible values are the same as documented on enum *atscmh_rs_code_mode*:

DTV_ATSCMH_SCCC_BLOCK_MODE

Series Concatenated Convolutional Code Block Mode.

Possible values are:

atscmh_sccc_block_mode

Table 1.173: enum atscmh_scc_block_mode

ID	Description
ATSCMH_SCCC_BLK_SEP	Separate SCCC: the SCCC outer code mode shall be set independently for each Group Region (A, B, C, D)
ATSCMH_SCCC_BLK_COMB	Combined SCCC: all four Regions shall have the same SCCC outer code mode.
ATSCMH_SCCC_BLK_RES	Reserved. Shouldn't be used.

DTV_ATSCMH_SCCC_CODE_MODE_A

Series Concatenated Convolutional Code Rate.

Possible values are:

atscmh_sccc_code_mode

Table 1.174: enum atscmh_sccc_code_mode

ID	Description
ATSCMH_SCCC_CODE_HLF	The outer code rate of a SCCC Block is 1/2 rate.
ATSCMH_SCCC_CODE_QTR	The outer code rate of a SCCC Block is 1/4 rate.
ATSCMH_SCCC_CODE_RES	to be documented.

DTV_ATSCMH_SCCC_CODE_MODE_B

Series Concatenated Convolutional Code Rate.

Possible values are the same as documented on enum *atscmh_sccc_code_mode*.

DTV_ATSCMH_SCCC_CODE_MODE_C

Series Concatenated Convolutional Code Rate.

Possible values are the same as documented on enum *atscmh_sccc_code_mode*.

DTV_ATSCMH_SCCC_CODE_MODE_D

Series Concatenated Convolutional Code Rate.

Possible values are the same as documented on enum *atscmh_sccc_code_mode*.

DTV_API_VERSION

Returns the major/minor version of the DVB API

DTV_CODE_RATE_HP

Used on terrestrial transmissions. The acceptable values are the ones described at *fe_transmit_mode*.

DTV_CODE_RATE_LP

Used on terrestrial transmissions. The acceptable values are the ones described at *fe_transmit_mode*.

DTV_GUARD_INTERVAL

Possible values are:

fe_guard_interval

Modulation guard interval

Table 1.175: enum fe_guard_interval

ID	Description
GUARD_INTERVAL_AUTO	Autodetect the guard interval
GUARD_INTERVAL_1_128	Guard interval 1/128
GUARD_INTERVAL_1_32	Guard interval 1/32
GUARD_INTERVAL_1_16	Guard interval 1/16
GUARD_INTERVAL_1_8	Guard interval 1/8
GUARD_INTERVAL_1_4	Guard interval 1/4
GUARD_INTERVAL_19_128	Guard interval 19/128
GUARD_INTERVAL_19_256	Guard interval 19/256
GUARD_INTERVAL_PN420	PN length 420 (1/4)
GUARD_INTERVAL_PN595	PN length 595 (1/6)
GUARD_INTERVAL_PN945	PN length 945 (1/9)

Notes:

1) If DTV_GUARD_INTERVAL is set the GUARD_INTERVAL_AUTO the hardware will try to find the correct guard interval (if capable) and will use TMCC to fill in the missing parameters.

2) Intervals 1/128, 19/128 and 19/256 are used only for DVB-T2 at present

3. DTMB specifies PN420, PN595 and PN945.

DTV_TRANSMISSION_MODE

Specifies the number of carriers used by the standard. This is used only on OFTM-based standards, e. g. DVB-T/T2, ISDB-T, DTMB

fe_transmit_mode

enum fe_transmit_mode: Number of carriers per channel

Table 1.176: enum fe_transmit_mode

ID	Description
TRANSMISSION_MODE_AUTO	Autodetect transmission mode. The hardware will try to find the correct FFT-size (if capable) to fill in the missing parameters.
TRANSMISSION_MODE_1K	Transmission mode 1K
TRANSMISSION_MODE_2K	Transmission mode 2K
TRANSMISSION_MODE_8K	Transmission mode 8K
TRANSMISSION_MODE_4K	Transmission mode 4K
TRANSMISSION_MODE_16K	Transmission mode 16K
TRANSMISSION_MODE_32K	Transmission mode 32K
TRANSMISSION_MODE_C1	Single Carrier (C=1) transmission mode (DTMB)
TRANSMISSION_MODE_C3780	Multi Carrier (C=3780) transmission mode (DTMB)

Notes:

1) ISDB-T supports three carrier/symbol-size: 8K, 4K, 2K. It is called 'mode' in the standard: Mode 1 is 2K, mode 2 is 4K, mode 3 is 8K

2) If DTV_TRANSMISSION_MODE is set the TRANSMISSION_MODE_AUTO the hardware will try to find the correct FFT-size (if capable) and will use TMCC to fill in the missing parameters.

3. DVB-T specifies 2K and 8K as valid sizes.
4. DVB-T2 specifies 1K, 2K, 4K, 8K, 16K and 32K.
5. DTMB specifies C1 and C3780.

DTV_HIERARCHY

Frontend hierarchy

fe_hierarchy

Frontend hierarchy

Table 1.177: enum fe_hierarchy

ID	Description
HIERARCHY_NONE	No hierarchy
HIERARCHY_AUTO	Autodetect hierarchy (if supported)
HIERARCHY_1	Hierarchy 1
HIERARCHY_2	Hierarchy 2
HIERARCHY_4	Hierarchy 4

DTV_STREAM_ID

DVB-S2, DVB-T2 and ISDB-S support the transmission of several streams on a single transport stream. This property enables the DVB driver to handle substream filtering, when supported by the hardware. By default, substream filtering is disabled.

For DVB-S2 and DVB-T2, the valid substream id range is from 0 to 255.

For ISDB, the valid substream id range is from 1 to 65535.

To disable it, you should use the special macro NO_STREAM_ID_FILTER.

Note: any value outside the id range also disables filtering.

DTV_DVBT2_PLP_ID_LEGACY

Obsolete, replaced with DTV_STREAM_ID.

DTV_ENUM_DELSYS

A Multi standard frontend needs to advertise the delivery systems provided. Applications need to enumerate the provided delivery systems, before using any other operation with the frontend. Prior to it's introduction, FE_GET_INFO was used to determine a frontend type. A frontend which provides more than a single delivery system, FE_GET_INFO doesn't help much. Applications which intends to use a multi-standard frontend must enumerate the delivery systems associated with it, rather than trying to use FE_GET_INFO. In the case of a legacy frontend, the result is just the same as with FE_GET_INFO, but in a more structured format

DTV_INTERLEAVING

Time interleaving to be used. Currently, used only on DTMB.

fe_interleaving

Table 1.178: enum fe_interleaving

ID	Description
INTERLEAVING_NONE	No interleaving.
INTERLEAVING_AUTO	Auto-detect interleaving.
INTERLEAVING_240	Interleaving of 240 symbols.
INTERLEAVING_720	Interleaving of 720 symbols.

DTV_LNA

Low-noise amplifier.

Hardware might offer controllable LNA which can be set manually using that parameter. Usually LNA could be found only from terrestrial devices if at all.

Possible values: 0, 1, LNA_AUTO

0, LNA off

1, LNA on

use the special macro LNA_AUTO to set LNA auto

Frontend statistics indicators

The values are returned via `dtv_property.stat`. If the property is supported, `dtv_property.stat.len` is bigger than zero.

For most delivery systems, `dtv_property.stat.len` will be 1 if the stats is supported, and the properties will return a single value for each parameter.

It should be noted, however, that new OFDM delivery systems like ISDB can use different modulation types for each group of carriers. On such standards, up to 3 groups of statistics can be provided, and `dtv_property.stat.len` is updated to reflect the "global" metrics, plus one metric per each carrier group (called "layer" on ISDB).

So, in order to be consistent with other delivery systems, the first value at *`dtv_property.stat.dtv_stats`* array refers to the global metric. The other elements of the array represent each layer, starting from layer A(index 1), layer B (index 2) and so on.

The number of filled elements are stored at `dtv_property.stat.len`.

Each element of the `dtv_property.stat.dtv_stats` array consists on two elements:

- `svalue` or `uvalue`, where `svalue` is for signed values of the measure (dB measures) and `uvalue` is for unsigned values (counters, relative scale)
- `scale` - Scale for the value. It can be:
  - `FE_SCALE_NOT_AVAILABLE` - The parameter is supported by the frontend, but it was not possible to collect it (could be a transitory or permanent condition)
  - `FE_SCALE_DECIBEL` - parameter is a signed value, measured in 1/1000 dB
  - `FE_SCALE_RELATIVE` - parameter is a unsigned value, where 0 means 0% and 65535 means 100%.
  - `FE_SCALE_COUNTER` - parameter is a unsigned value that counts the occurrence of an event, like bit error, block error, or lapsed time.

DTV_STAT_SIGNAL_STRENGTH

Indicates the signal strength level at the analog part of the tuner or of the demod.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_DECIBEL` - signal strength is in 0.001 dBm units, power measured in miliwatts. This value is generally negative.
- `FE_SCALE_RELATIVE` - The frontend provides a 0% to 100% measurement for power (actually, 0 to 65535).

DTV_STAT_CNR

Indicates the Signal to Noise ratio for the main carrier.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_DECIBEL` - Signal/Noise ratio is in 0.001 dB units.
- `FE_SCALE_RELATIVE` - The frontend provides a 0% to 100% measurement for Signal/Noise (actually, 0 to 65535).

DTV_STAT_PRE_ERROR_BIT_COUNT

Measures the number of bit errors before the forward error correction (FEC) on the inner coding block (before Viterbi, LDPC or other inner code).

This measure is taken during the same interval as DTV_STAT_PRE_TOTAL_BIT_COUNT.

In order to get the BER (Bit Error Rate) measurement, it should be divided by *DTV_STAT_PRE_TOTAL_BIT_COUNT* .

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- FE_SCALE_NOT_AVAILABLE - it failed to measure it, or the measurement was not complete yet.
- FE_SCALE_COUNTER - Number of error bits counted before the inner coding.

DTV_STAT_PRE_TOTAL_BIT_COUNT

Measures the amount of bits received before the inner code block, during the same period as *DTV_STAT_PRE_ERROR_BIT_COUNT* measurement was taken.

It should be noted that this measurement can be smaller than the total amount of bits on the transport stream, as the frontend may need to manually restart the measurement, losing some data between each measurement interval.

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- FE_SCALE_NOT_AVAILABLE - it failed to measure it, or the measurement was not complete yet.
- FE_SCALE_COUNTER - Number of bits counted while measuring *DTV_STAT_PRE_ERROR_BIT_COUNT* .

DTV_STAT_POST_ERROR_BIT_COUNT

Measures the number of bit errors after the forward error correction (FEC) done by inner code block (after Viterbi, LDPC or other inner code).

This measure is taken during the same interval as DTV_STAT_POST_TOTAL_BIT_COUNT.

In order to get the BER (Bit Error Rate) measurement, it should be divided by *DTV_STAT_POST_TOTAL_BIT_COUNT* .

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- FE_SCALE_NOT_AVAILABLE - it failed to measure it, or the measurement was not complete yet.
- FE_SCALE_COUNTER - Number of error bits counted after the inner coding.

DTV_STAT_POST_TOTAL_BIT_COUNT

Measures the amount of bits received after the inner coding, during the same period as *DTV_STAT_POST_ERROR_BIT_COUNT* measurement was taken.

It should be noted that this measurement can be smaller than the total amount of bits on the transport stream, as the frontend may need to manually restart the measurement, losing some data between each measurement interval.

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- FE_SCALE_NOT_AVAILABLE - it failed to measure it, or the measurement was not complete yet.
- FE_SCALE_COUNTER - Number of bits counted while measuring *DTV_STAT_POST_ERROR_BIT_COUNT* .

DTV_STAT_ERROR_BLOCK_COUNT

Measures the number of block errors after the outer forward error correction coding (after Reed-Solomon or other outer code).

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- FE_SCALE_NOT_AVAILABLE - it failed to measure it, or the measurement was not complete yet.
- FE_SCALE_COUNTER - Number of error blocks counted after the outer coding.

DTV-STAT_TOTAL_BLOCK_COUNT

Measures the total number of blocks received during the same period as *DTV_STAT_ERROR_BLOCK_COUNT* measurement was taken.

It can be used to calculate the PER indicator, by dividing *DTV_STAT_ERROR_BLOCK_COUNT* by *DTV-STAT_TOTAL_BLOCK_COUNT* .

Possible scales for this metric are:

- FE_SCALE_NOT_AVAILABLE - it failed to measure it, or the measurement was not complete yet.
- FE_SCALE_COUNTER - Number of blocks counted while measuring *DTV_STAT_ERROR_BLOCK_COUNT* .

Properties used on terrestrial delivery systems

DVB-T delivery system

The following parameters are valid for DVB-T:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_CODE_RATE_HP*
- *DTV_CODE_RATE_LP*
- *DTV_GUARD_INTERVAL*

- *DTV_TRANSMISSION_MODE*
- *DTV_HIERARCHY*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

DVB-T2 delivery system

DVB-T2 support is currently in the early stages of development, so expect that this section maygrow and become more detailed with time.

The following parameters are valid for DVB-T2:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_CODE_RATE_HP*
- *DTV_CODE_RATE_LP*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_HIERARCHY*
- *DTV_STREAM_ID*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

ISDB-T delivery system

This ISDB-T/ISDB-Tsb API extension should reflect all information needed to tune any ISDB-T/ISDB-Tsb hardware. Of course it is possible that some very sophisticated devices won't need certain parameters to tune.

The information given here should help application writers to know how to handle ISDB-T and ISDB-Tsb hardware using the Linux DVB-API.

The details given here about ISDB-T and ISDB-Tsb are just enough to basically show the dependencies between the needed parameter values, but surely some information is left out. For more detailed information see the following documents:

ARIB STD-B31 - "Transmission System for Digital Terrestrial Television Broadcasting" and

ARIB TR-B14 - "Operational Guidelines for Digital Terrestrial Television Broadcasting".

In order to understand the ISDB specific parameters, one has to have some knowledge the channel structure in ISDB-T and ISDB-Tsb. I.e. it has to be known to the reader that an ISDB-T channel consists of 13 segments, that it can have up to 3 layer sharing those segments, and things like that.

The following parameters are valid for ISDB-T:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_ISDBT_LAYER_ENABLED*
- *DTV_ISDBT_PARTIAL_RECEPTION*
- *DTV_ISDBT_SOUND_BROADCASTING*
- *DTV_ISDBT_SB_SUBCHANNEL_ID*
- *DTV_ISDBT_SB_SEGMENT_IDX*
- *DTV_ISDBT_SB_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERA_FEC*
- *DTV_ISDBT_LAYERA_MODULATION*
- *DTV_ISDBT_LAYERA_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERA_TIME_INTERLEAVING*
- *DTV_ISDBT_LAYERB_FEC*
- *DTV_ISDBT_LAYERB_MODULATION*
- *DTV_ISDBT_LAYERB_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERB_TIME_INTERLEAVING*
- *DTV_ISDBT_LAYERC_FEC*
- *DTV_ISDBT_LAYERC_MODULATION*
- *DTV_ISDBT_LAYERC_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERC_TIME_INTERLEAVING*

In addition, the *DTV QoS statistics* are also valid.

ATSC delivery system

The following parameters are valid for ATSC:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*

In addition, the *DTV QoS statistics* are also valid.

ATSC-MH delivery system

The following parameters are valid for ATSC-MH:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_BANDWIDTH_HZ*
- *DTV_ATSCMH_FIC_VER*
- *DTV_ATSCMH_PARADE_ID*
- *DTV_ATSCMH_NOG*
- *DTV_ATSCMH_TNOG*
- *DTV_ATSCMH_SGN*
- *DTV_ATSCMH_PRC*
- *DTV_ATSCMH_RS_FRAME_MODE*
- *DTV_ATSCMH_RS_FRAME_ENSEMBLE*
- *DTV_ATSCMH_RS_CODE_MODE_PRI*
- *DTV_ATSCMH_RS_CODE_MODE_SEC*
- *DTV_ATSCMH_SCCC_BLOCK_MODE*
- *DTV_ATSCMH_SCCC_CODE_MODE_A*
- *DTV_ATSCMH_SCCC_CODE_MODE_B*
- *DTV_ATSCMH_SCCC_CODE_MODE_C*
- *DTV_ATSCMH_SCCC_CODE_MODE_D*

In addition, the *DTV QoS statistics* are also valid.

DTMB delivery system

The following parameters are valid for DTMB:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_INNER_FEC*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*

- *DTV_INTERLEAVING*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

Properties used on cable delivery systems

DVB-C delivery system

The DVB-C Annex-A is the widely used cable standard. Transmission uses QAM modulation.

The DVB-C Annex-C is optimized for 6MHz, and is used in Japan. It supports a subset of the Annex A modulation types, and a roll-off of 0.13, instead of 0.15

The following parameters are valid for DVB-C Annex A/C:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_INVERSION*
- *DTV_SYMBOL_RATE*
- *DTV_INNER_FEC*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

DVB-C Annex B delivery system

The DVB-C Annex-B is only used on a few Countries like the United States.

The following parameters are valid for DVB-C Annex B:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_INVERSION*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

Properties used on satellite delivery systems

DVB-S delivery system

The following parameters are valid for DVB-S:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_INVERSION*
- *DTV_SYMBOL_RATE*
- *DTV_INNER_FEC*
- *DTV_VOLTAGE*
- *DTV_TONE*

In addition, the *DTV QoS statistics* are also valid.

Future implementations might add those two missing parameters:

- *DTV_DISEQC_MASTER*
- *DTV_DISEQC_SLAVE_REPLY*

DVB-S2 delivery system

In addition to all parameters valid for DVB-S, DVB-S2 supports the following parameters:

- *DTV_MODULATION*
- *DTV_PILOT*
- *DTV_ROLLOFF*
- *DTV_STREAM_ID*

In addition, the *DTV QoS statistics* are also valid.

Turbo code delivery system

In addition to all parameters valid for DVB-S, turbo code supports the following parameters:

- *DTV_MODULATION*

ISDB-S delivery system

The following parameters are valid for ISDB-S:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*

- *DTV_INVERSION*
- *DTV_SYMBOL_RATE*
- *DTV_INNER_FEC*
- *DTV_VOLTAGE*
- *DTV_STREAM_ID*

Frontend Function Calls

DVB frontend open()

Name

fe-open - Open a frontend device

Synopsis

```
#include <fcntl.h>
```

int **open**(const char *_device_name_, int _flags_)

Arguments

device_name Device to be opened.

flags Open flags. Access can either be O_RDWR or O_RDONLY.

Multiple opens are allowed with O_RDONLY. In this mode, only query and read ioctls are allowed.

Only one open is allowed in O_RDWR. In this mode, all ioctls are allowed.

When the O_NONBLOCK flag is given, the system calls may return EAGAIN error code when no data is available or when the device driver is temporarily busy.

Other flags have no effect.

Description

This system call opens a named frontend device (/dev/dvb/adapter?/frontend?) for subsequent use. Usually the first thing to do after a successful open is to find out the frontend type with *ioctl FE_GET_INFO* .

The device can be opened in read-only mode, which only allows monitoring of device status and statistics, or read/write mode, which allows any kind of use (e.g. performing tuning operations.)

In a system with multiple front-ends, it is usually the case that multiple devices cannot be open in read/write mode simultaneously. As long as a front-end device is opened in read/write mode, other open() calls in read/write mode will either fail or block, depending on whether non-blocking or blocking mode was specified. A front-end device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the F_SETFL command of the fcntl system call. This is a standard system call, documented in the Linux manual page for fcntl. When an open() call has succeeded, the device will be ready for use in the specified mode. This implies that the corresponding hardware is powered up, and that other front-ends may have been powered down to make that possible.

Return Value

On success *open()* returns the new file descriptor. On error, -1 is returned, and the `errno` variable is set appropriately.

Possible error codes are:

EACCES The caller has no permission to access the device.

EBUSY The the device driver is already in use.

ENXIO No device corresponding to this device special file exists.

ENOMEM Not enough kernel memory was available to complete the request.

EMFILE The process already has the maximum number of files open.

ENFILE The limit on the total number of files open on the system has been reached.

ENODEV The device got removed.

DVB frontend close()

Name

fe-close - Close a frontend device

Synopsis

```
#include <unistd.h>
```

int **close**(int *fd*)

Arguments

fd File descriptor returned by *open()*.

Description

This system call closes a previously opened front-end device. After closing a front-end device, its corresponding hardware might be powered down automatically.

Return Value

The function returns 0 on success, -1 on failure and the `errno` is set appropriately. Possible error codes:

EBADF `fd` is not a valid open file descriptor.

ioctl FE_GET_INFO

Name

FE_GET_INFO - Query DVB frontend capabilities and returns information about the - front-end. This call only requires read-only access to the device

Synopsis

int **ioctl**(int *fd*, *FE_GET_INFO*, struct *dvb_frontend_info* *argp)

Arguments

fd File descriptor returned by *open()* .

argp pointer to struct struct *dvb_frontend_info*

Description

All DVB frontend devices support the FE_GET_INFO ioctl. It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities. The ioctl takes a pointer to dvb_frontend_info which is filled by the driver. When the driver is not compatible with this specification the ioctl returns an error.

dvb_frontend_info

Table 1.179: struct dvb_frontend_info

char	name[128]	Name of the frontend
fe_type_t	type	**DEPRECATED**. DVBv3 type. Should not be used on modern programs, as a frontend may have more than one type. So, the DVBv5 API should be used instead to enumerate and select the frontend type.
uint32_t	frequency_min	Minimal frequency supported by the frontend
uint32_t	frequency_max	Maximal frequency supported by the frontend
uint32_t	frequency_stepsize	Frequency step - all frequencies are multiple of this value
uint32_t	frequency_tolerance	Tolerance of the frequency
uint32_t	symbol_rate_min	Minimal symbol rate (for Cable/Satellite systems), in bauds
uint32_t	symbol_rate_max	Maximal symbol rate (for Cable/Satellite systems), in bauds
uint32_t	symbol_rate_tolerance	Maximal symbol rate tolerance, in ppm
uint32_t	notifier_delay	**DEPRECATED**. Not used by any driver.
enum *fe_caps*	caps	Capabilities supported by the frontend

Note:

The frequencies are specified in Hz for Terrestrial and Cable systems. They're specified in kHz for Satellite systems

frontend capabilities

Capabilities describe what a frontend can do. Some capabilities are supported only on some specific frontend types.

fe_caps

Table 1.180: enum fe_caps

ID	Description
FE_IS_STUPID	There's something wrong at the frontend, and it can't report its capabilities
FE_CAN_INVERSION_AUTO	The frontend is capable of auto-detecting inversion
FE_CAN_FEC_1_2	The frontend supports FEC 1/2
FE_CAN_FEC_2_3	The frontend supports FEC 2/3
FE_CAN_FEC_3_4	The frontend supports FEC 3/4
FE_CAN_FEC_4_5	The frontend supports FEC 4/5
FE_CAN_FEC_5_6	The frontend supports FEC 5/6
FE_CAN_FEC_6_7	The frontend supports FEC 6/7
FE_CAN_FEC_7_8	The frontend supports FEC 7/8
FE_CAN_FEC_8_9	The frontend supports FEC 8/9
FE_CAN_FEC_AUTO	The frontend can autodetect FEC.
FE_CAN_QPSK	The frontend supports QPSK modulation
FE_CAN_QAM_16	The frontend supports 16-QAM modulation
FE_CAN_QAM_32	The frontend supports 32-QAM modulation
FE_CAN_QAM_64	The frontend supports 64-QAM modulation
FE_CAN_QAM_128	The frontend supports 128-QAM modulation
FE_CAN_QAM_256	The frontend supports 256-QAM modulation
FE_CAN_QAM_AUTO	The frontend can autodetect modulation
FE_CAN_TRANSMISSION_MODE_AUTO	The frontend can autodetect the transmission mode
FE_CAN_BANDWIDTH_AUTO	The frontend can autodetect the bandwidth
FE_CAN_GUARD_INTERVAL_AUTO	The frontend can autodetect the guard interval
FE_CAN_HIERARCHY_AUTO	The frontend can autodetect hierarch
FE_CAN_8VSB	The frontend supports 8-VSB modulation
FE_CAN_16VSB	The frontend supports 16-VSB modulation
FE_HAS_EXTENDED_CAPS	Currently, unused
FE_CAN_MULTISTREAM	The frontend supports multistream filtering
FE_CAN_TURBO_FEC	The frontend supports turbo FEC modulation
FE_CAN_2G_MODULATION	The frontend supports "2nd generation modulation" (DVB-S2/T2)>
FE_NEEDS_BENDING	Not supported anymore, don't use it
FE_CAN_RECOVER	The frontend can recover from a cable unplug automatically
FE_CAN_MUTE_TS	The frontend can stop spurious TS data output

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_READ_STATUS

Name

FE_READ_STATUS - Returns status information about the front-end. This call only requires - read-only access to the device

Synopsis

int **ioctl**(int *fd*, *FE_READ_STATUS*, unsigned int **status*)

Arguments

fd File descriptor returned by *open()* .

status pointer to a bitmask integer filled with the values defined by enum *fe_status*.

Description

All DVB frontend devices support the FE_READ_STATUS ioctl. It is used to check about the locking status of the frontend after being tuned. The ioctl takes a pointer to an integer where the status will be written.

Note:

The size of status is actually sizeof(enum fe_status), with varies according with the architecture. This needs to be fixed in the future.

int fe_status

The fe_status parameter is used to indicate the current state and/or state changes of the frontend hardware. It is produced using the enum *fe_status* values on a bitmask

fe_status

Table 1.181: enum fe_status

ID	Description
FE_HAS_SIGNAL	The frontend has found something above the noise level
FE_HAS_CARRIER	The frontend has found a DVB signal
FE_HAS_VITERBI	The frontend FEC inner coding (Viterbi, LDPC or other inner code) is stable
FE_HAS_SYNC	Synchronization bytes was found
FE_HAS_LOCK	The DVB were locked and everything is working
FE_TIMEDOUT	no lock within the last about 2 seconds
FE_REINIT	The frontend was reinitialized, application is recommended to reset DiSEqC, tone and parameters

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_PROPERTY, FE_GET_PROPERTY

Name

FE_SET_PROPERTY - FE_GET_PROPERTY - FE_SET_PROPERTY sets one or more frontend properties. - FE_GET_PROPERTY returns one or more frontend properties.

Synopsis

int **ioctl**(int *fd*, *FE_GET_PROPERTY*, struct *dtv_properties* *argp)

int **ioctl**(int *fd*, *FE_SET_PROPERTY*, struct *dtv_properties* *argp)

Arguments

fd File descriptor returned by *open()* .

argp pointer to struct *dtv_properties*

Description

All DVB frontend devices support the FE_SET_PR0PERTY and FE_GET_PR0PERTY ioctls. The supported properties and statistics depends on the delivery system and on the device:

- FE_SET_PR0PERTY:
  - This ioctl is used to set one or more frontend properties.
  - This is the basic command to request the frontend to tune into some frequency and to start decoding the digital TV signal.
  - This call requires read/write access to the device.
  - At return, the values are updated to reflect the actual parameters used.
- FE_GET_PR0PERTY:
  - This ioctl is used to get properties and statistics from the frontend.
  - No properties are changed, and statistics aren't reset.
  - This call only requires read-only access to the device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_RESET_OVERLOAD

Name

FE_DISEQC_RESET_OVERLOAD - Restores the power to the antenna subsystem, if it was powered off due - to power overload.

Synopsis

int **ioctl**(int *fd*, *FE_DISEQC_RESET_OVERLOAD*, NULL)

Arguments

fd File descriptor returned by *open()* .

Description

If the bus has been automatically powered off due to power overload, this ioctl call restores the power to the bus. The call requires read/write access to the device. This call has no effect if the device is manually powered off. Not all DVB adapters support this ioctl.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_SEND_MASTER_CMD

Name

FE_DISEQC_SEND_MASTER_CMD - Sends a DiSEqC command

Synopsis

int **ioctl**(int *fd*, *FE_DISEQC_SEND_MASTER_CMD*, struct *dvb_diseqc_master_cmd* **argp*)

Arguments

fd File descriptor returned by *open()* .

argp pointer to struct *dvb_diseqc_master_cmd*

Description

Sends a DiSEqC command to the antenna subsystem.

dvb_diseqc_master_cmd

Table 1.182: struct dvb_diseqc_master_cmd

uint8_t	msg[6]	DiSEqC message (framing, address, command, data[3])
uint8_t	msg_len	Length of the DiSEqC message. Valid values are 3 to 6

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_RECV_SLAVE_REPLY

Name

FE_DISEQC_RECV_SLAVE_REPLY - Receives reply from a DiSEqC 2.0 command

Synopsis

int **ioctl**(int *fd*, *FE_DISEQC_RECV_SLAVE_REPLY*, struct *dvb_diseqc_slave_reply* *argp)

Arguments

fd File descriptor returned by *open()* .

argp pointer to struct *dvb_diseqc_slave_reply*

Description

Receives reply from a DiSEqC 2.0 command.

dvb_diseqc_slave_reply

Table 1.183: struct dvb_diseqc_slave_reply

uint8_t	msg[4]	DiSEqC message (framing, data[3])
uint8_t	msg_len	Length of the DiSEqC message. Valid values are 0 to 4, where 0 means no msg
int	timeout	Return from ioctl after timeout ms with errorcode when no message was received

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_SEND_BURST

Name

FE_DISEQC_SEND_BURST - Sends a 22KHz tone burst for 2x1 mini DiSEqC satellite selection.

Synopsis

int **ioctl**(int *fd*, *FE_DISEQC_SEND_BURST*, enum *fe_sec_mini_cmd* *tone)

Arguments

fd File descriptor returned by *open()* .

tone pointer to enum *fe_sec_mini_cmd*

Description

This ioctl is used to set the generation of a 22kHz tone burst for mini DiSEqC satellite selection for 2x1 switches. This call requires read/write permissions.

It provides support for what's specified at Digital Satellite Equipment Control (DiSEqC) - Simple "ToneBurst" Detection Circuit specification.

fe_sec_mini_cmd

Table 1.184: enum fe_sec_mini_cmd

ID	Description
SEC_MINI_A	Sends a mini-DiSEqC 22kHz '0' Tone Burst to select satellite-A
SEC_MINI_B	Sends a mini-DiSEqC 22kHz '1' Data Burst to select satellite-B

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_TONE

Name

FE_SET_TONE - Sets/resets the generation of the continuous 22kHz tone.

Synopsis

int **ioctl**(int *fd*, *FE_SET_TONE*, enum *fe_sec_tone_mode* **tone*)

Arguments

fd File descriptor returned by *open()* .

tone pointer to enum *fe_sec_tone_mode*

Description

This ioctl is used to set the generation of the continuous 22kHz tone. This call requires read/write permissions.

Usually, satellite antenna subsystems require that the digital TV device to send a 22kHz tone in order to select between high/low band on some dual-band LNBf. It is also used to send signals to DiSEqC equipment, but this is done using the DiSEqC ioctls.

Attention:

If more than one device is connected to the same antenna, setting a tone may interfere on other devices, as they may lose the capability of selecting the band. So, it is recommended that applications would change to SEC_TONE_OFF when the device is not used.

fe_sec_tone_mode

Table 1.185: enum fe_sec_tone_mode

ID	Description
SEC_TONE_ON	Sends a 22kHz tone burst to the antenna
SEC_TONE_OFF	Don't send a 22kHz tone to the antenna (except if the FE_DISEQC_* ioctls are called)

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_VOLTAGE

Name

FE_SET_VOLTAGE - Allow setting the DC level sent to the antenna subsystem.

Synopsis

int **ioctl**(int *fd*, *FE_SET_VOLTAGE*, enum *fe_sec_voltage* **voltage*)

Arguments

fd File descriptor returned by *open()* .

voltage pointer to enum *fe_sec_voltage*

Valid values are described at enum *fe_sec_voltage*.

Description

This ioctl allows to set the DC voltage level sent through the antenna cable to 13V, 18V or off.

Usually, a satellite antenna subsystems require that the digital TV device to send a DC voltage to feed power to the LNBf. Depending on the LNBf type, the polarization or the intermediate frequency (IF) of the LNBf can controlled by the voltage level. Other devices (for example, the ones that implement DISEqC and multipoint LNBf's don't need to control the voltage level, provided that either 13V or 18V is sent to power up the LNBf.

Attention:

if more than one device is connected to the same antenna, setting a voltage level may interfere on other devices, as they may lose the capability of setting polarization or IF. So, on those cases, setting the voltage to SEC_VOLTAGE_OFF while the device is not is used is recommended.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_ENABLE_HIGH_LNB_VOLTAGE

Name

FE_ENABLE_HIGH_LNB_VOLTAGE - Select output DC level between normal LNBf voltages or higher LNBf - voltages.

Synopsis

int **ioctl**(int *fd*, *FE_ENABLE_HIGH_LNB_VOLTAGE*, unsigned int *high*)

Arguments

fd File descriptor returned by *open()* .

high Valid flags:

- 0 - normal 13V and 18V.
- >0 - enables slightly higher voltages instead of 13/18V, in order to compensate for long antenna cables.

Description

Select output DC level between normal LNBf voltages or higher LNBf voltages between 0 (normal) or a value grater than 0 for higher voltages.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_FRONTEND_TUNE_MODE

Name

FE_SET_FRONTEND_TUNE_MODE - Allow setting tuner mode flags to the frontend.

Synopsis

int **ioctl**(int *fd*, *FE_SET_FRONTEND_TUNE_MODE*, unsigned int *flags*)

Arguments

fd File descriptor returned by *open()* .

flags Valid flags:

- 0 - normal tune mode

- FE_TUNE_MODE_ONESHOT - When set, this flag will disable any zigzagging or other “normal” tuning behaviour. Additionally, there will be no automatic monitoring of the lock status, and hence no frontend events will be generated. If a frontend device is closed, this flag will be automatically turned off when the device is reopened read-write.

Description

Allow setting tuner mode flags to the frontend, between 0 (normal) or FE_TUNE_MODE_ONESHOT mode

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

DVB Frontend legacy API (a. k. a. DVBv3)

The usage of this API is deprecated, as it doesn’t support all digital TV standards, doesn’t provide good statistics measurements and provides incomplete information. This is kept only to support legacy applications.

Frontend Legacy Data Types

Frontend type

For historical reasons, frontend types are named by the type of modulation used in transmission. The fontend types are given by fe_type_t type, defined as:

fe_type

Table 1.186: Frontend types

fe_type	Description	*DTV_DELIVERY_SYSTEM* equivalent type
FE_QPSK	For DVB-S standard	SYS_DVBS
FE_QAM	For DVB-C annex A standard	SYS_DVBC_ANNEX_A
FE_OFDM	For DVB-T standard	SYS_DVBT
FE_ATSC	For ATSC standard (terrestrial) or for DVB-C Annex B (cable) used in US.	SYS_ATSC (terrestrial) or SYS_DVBC_ANNEX_B (cable)

Newer formats like DVB-S2, ISDB-T, ISDB-S and DVB-T2 are not described at the above, as they’re supported via the new *FE_GET_PROPERTY/FE_GET_SET_PROPERTY* ioctl’s, using the *DTV_DELIVERY_SYSTEM* parameter.

In the old days, struct *dvb_frontend_info* used to contain `fe_type_t` field to indicate the delivery systems, filled with either FE_QPSK, FE_QAM, FE_OFDM or FE_ATSC. While this is still filled to keep backward compatibility, the usage of this field is deprecated, as it can report just one delivery system, but some devices support multiple delivery systems. Please use *DTV_ENUM_DELSYS* instead.

On devices that support multiple delivery systems, struct *dvb_frontend_info*::fe_type_t is filled with the currently standard, as selected by the last call to *FE_SET_PROPERTY* using the *DTV_DELIVERY_SYSTEM* property.

Frontend bandwidth

fe_bandwidth

Table 1.187: enum fe_bandwidth

ID	Description
BANDWIDTH_AUTO	Autodetect bandwidth (if supported)
BANDWIDTH_1_712_MHZ	1.712 MHz
BANDWIDTH_5_MHZ	5 MHz
BANDWIDTH_6_MHZ	6 MHz
BANDWIDTH_7_MHZ	7 MHz
BANDWIDTH_8_MHZ	8 MHz
BANDWIDTH_10_MHZ	10 MHz

dvb_frontend_parameters

frontend parameters

The kind of parameters passed to the frontend device for tuning depend on the kind of hardware you are using.

The struct dvb_frontend_parameters uses an union with specific per-system parameters. However, as newer delivery systems required more data, the structure size weren't enough to fit, and just extending its size would break the existing applications. So, those parameters were replaced by the usage of *FE_GET_PROPERTY/FE_SET_PROPERTY* ioctl's. The new API is flexible enough to add new parameters to existing delivery systems, and to add newer delivery systems.

So, newer applications should use *FE_GET_PROPERTY/FE_SET_PROPERTY* instead, in order to be able to support the newer System Delivery like DVB-S2, DVB-T2, DVB-C2, ISDB, etc.

All kinds of parameters are combined as an union in the FrontendParameters structure:

```
struct dvb_frontend_parameters {
    uint32_t frequency;     /* (absolute) frequency in Hz for QAM/OFDM */
                /* intermediate frequency in kHz for QPSK */
    fe_spectral_inversion_t inversion;
    union {
        struct dvb_qpsk_parameters qpsk;
        struct dvb_qam_parameters  qam;
        struct dvb_ofdm_parameters ofdm;
        struct dvb_vsb_parameters  vsb;
    } u;
};
```

In the case of QPSK frontends the frequency field specifies the intermediate frequency, i.e. the offset which is effectively added to the local oscillator frequency (LOF) of the LNB. The intermediate frequency has to be specified in units of kHz. For QAM and OFDM frontends the frequency specifies the absolute frequency and is given in Hz.

dvb_qpsk_parameters

QPSK parameters

For satellite QPSK frontends you have to use the dvb_qpsk_parameters structure:

```
struct dvb_qpsk_parameters {
    uint32_t        symbol_rate;  /* symbol rate in Symbols per second */
    fe_code_rate_t  fec_inner;    /* forward error correction (see above) */
};
```

dvb_qam_parameters

QAM parameters

for cable QAM frontend you use the dvb_qam_parameters structure:

```
struct dvb_qam_parameters {
    uint32_t         symbol_rate; /* symbol rate in Symbols per second */
    fe_code_rate_t   fec_inner;   /* forward error correction (see above) */
    fe_modulation_t  modulation;  /* modulation type (see above) */
};
```

dvb_vsb_parameters

VSB parameters

ATSC frontends are supported by the dvb_vsb_parameters structure:

```
struct dvb_vsb_parameters {
    fe_modulation_t modulation; /* modulation type (see above) */
};
```

dvb_ofdm_parameters

OFDM parameters

DVB-T frontends are supported by the dvb_ofdm_parameters structure:

```
struct dvb_ofdm_parameters {
    fe_bandwidth_t      bandwidth;
    fe_code_rate_t      code_rate_HP;  /* high priority stream code rate */
    fe_code_rate_t      code_rate_LP;  /* low priority stream code rate */
    fe_modulation_t     constellation; /* modulation type (see above) */
    fe_transmit_mode_t  transmission_mode;
    fe_guard_interval_t guard_interval;
    fe_hierarchy_t      hierarchy_information;
};
```

dvb_frontend_event

frontend events

```
struct dvb_frontend_event {
    fe_status_t status;
    struct dvb_frontend_parameters parameters;
};
```

Frontend Legacy Function Calls

Those functions are defined at DVB version 3. The support is kept in the kernel due to compatibility issues only. Their usage is strongly not recommended

FE_READ_BER

Name

FE_READ_BER

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *FE_READ_BER*, uint32_t **ber*)

Arguments

fd File descriptor returned by *open()*.

ber The bit error rate is stored into *ber.

Description

This ioctl call returns the bit error rate for the signal currently received/demodulated by the front-end. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

FE_READ_SNR

Name

FE_READ_SNR

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *FE_READ_SNR*, int16_t **snr*)

Arguments

fd File descriptor returned by *open()*.

snr The signal-to-noise ratio is stored into *snr.

Description

This ioctl call returns the signal-to-noise ratio for the signal currently received by the front-end. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

FE_READ_SIGNAL_STRENGTH

Name

FE_READ_SIGNAL_STRENGTH

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *FE_READ_SIGNAL_STRENGTH*, uint16_t **strength*)

Arguments

fd File descriptor returned by *open()*.

strength The signal strength value is stored into *strength.

Description

This ioctl call returns the signal strength value for the signal currently received by the front-end. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

FE_READ_UNCORRECTED_BLOCKS

Name

FE_READ_UNCORRECTED_BLOCKS

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *FE_READ_UNCORRECTED_BLOCKS*, uint32_t **ublocks*)

Arguments

fd File descriptor returned by *open()*.

ublocks The total number of uncorrected blocks seen by the driver so far.

Description

This ioctl call returns the number of uncorrected blocks detected by the device driver during its lifetime. For meaningful measurements, the increment in block count during a specific time interval should be calculated. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

FE_SET_FRONTEND

Attention:

This ioctl is deprecated.

Name

FE_SET_FRONTEND

Synopsis

int **ioctl**(int *fd*, *FE_SET_FRONTEND*, struct *dvb_frontend_parameters *p*)

Arguments

fd File descriptor returned by *open()*.

p Points to parameters for tuning operation.

Description

This ioctl call starts a tuning operation using specified parameters. The result of this call will be successful if the parameters were valid and the tuning could be initiated. The result of the tuning operation in itself, however, will arrive asynchronously as an event (see documentation for *FE_GET_EVENT* and FrontendEvent.) If a new *FE_SET_FRONTEND* operation is initiated before the previous one was completed, the previous operation will be aborted in favor of the new one. This command requires read/write access to the device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	Maximum supported symbol rate reached.

FE_GET_FRONTEND

Name

FE_GET_FRONTEND

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *FE_GET_FRONTEND*, struct *dvb_frontend_parameters* *p)

Arguments

fd File descriptor returned by *open()*.

p Points to parameters for tuning operation.

Description

This ioctl call queries the currently effective frontend parameters. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	Maximum supported symbol rate reached.

FE_GET_EVENT

Name

FE_GET_EVENT

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *FE_GET_EVENT*, struct *dvb_frontend_event* *ev)

Arguments

fd File descriptor returned by *open()*.

ev Points to the location where the event, if any, is to be stored.

Description

This ioctl call returns a frontend event if available. If an event is not available, the behavior depends on whether the device is in blocking or non-blocking mode. In the latter case, the call fails immediately with errno set to EWOULDBLOCK. In the former case, the call blocks until an event becomes available.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EWOULDBLOCK	There is no event pending, and the device is in non-blocking mode.
EOVERFLOW	Overflow in event queue - one or more events were lost.

FE_DISHNETWORK_SEND_LEGACY_CMD

Name

FE_DISHNETWORK_SEND_LEGACY_CMD

Synopsis

int **ioctl**(int *fd*, *FE_DISHNETWORK_SEND_LEGACY_CMD*, unsigned long *cmd*)

Arguments

fd File descriptor returned by *open()*.

cmd Sends the specified raw cmd to the dish via DISEqC.

Description

Warning:

This is a very obscure legacy command, used only at stv0299 driver. Should not be used on newer drivers.

It provides a non-standard method for selecting Diseqc voltage on the frontend, for Dish Network legacy switches.

As support for this ioctl were added in 2004, this means that such dishes were already legacy in 2004.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

1.3.3 DVB Demux Device

The DVB demux device controls the filters of the DVB hardware/software. It can be accessed through `/dev/adapter?/demux?`. Data types and and ioctl definitions can be accessed by including `linux/dvb/dmx.h` in your application.

Demux Data Types

Output for the demux

dmx_output

Table 1.188: enum dmx_output

ID	Description
DMX_OUT_DECODER	Streaming directly to decoder.
DMX_OUT_TAP	Output going to a memory buffer (to be retrieved via the read command). Delivers the stream output to the demux device on which the ioctl is called.
DMX_OUT_TS_TAP	Output multiplexed into a new TS (to be retrieved by reading from the logical DVR device). Routes output to the logical DVR device `/dev/dvb/adapter?/dvr?`, which delivers a TS multiplexed from all filters for which `DMX_OUT_TS_TAP` was specified.
DMX_OUT_TSDEMUX_TAP	Like *DMX_OUT_TS_TAP* but retrieved from the DMX device.

dmx_input_t

dmx_input

```
typedef enum
{
    DMX_IN_FRONTEND, /* Input from a front-end device.  */
    DMX_IN_DVR       /* Input from the logical DVR device.  */
} dmx_input_t;
```

dmx_pes_type_t

dmx_pes_type

```
typedef enum
{
    DMX_PES_AUDIO0,
    DMX_PES_VIDEO0,
    DMX_PES_TELETEXT0,
    DMX_PES_SUBTITLE0,
    DMX_PES_PCR0,

    DMX_PES_AUDIO1,
    DMX_PES_VIDEO1,
    DMX_PES_TELETEXT1,
    DMX_PES_SUBTITLE1,
    DMX_PES_PCR1,

    DMX_PES_AUDIO2,
    DMX_PES_VIDEO2,
    DMX_PES_TELETEXT2,
    DMX_PES_SUBTITLE2,
    DMX_PES_PCR2,

    DMX_PES_AUDIO3,
    DMX_PES_VIDEO3,
    DMX_PES_TELETEXT3,
    DMX_PES_SUBTITLE3,
    DMX_PES_PCR3,

    DMX_PES_OTHER
} dmx_pes_type_t;
```

struct dmx_filter

dmx_filter

```
 typedef struct dmx_filter
{
    __u8  filter[DMX_FILTER_SIZE];
    __u8  mask[DMX_FILTER_SIZE];
    __u8  mode[DMX_FILTER_SIZE];
} dmx_filter_t;
```

dmx_sct_filter_params

struct dmx_sct_filter_params

```
struct dmx_sct_filter_params
{
    __u16          pid;
    dmx_filter_t   filter;
    __u32          timeout;
    __u32          flags;
#define DMX_CHECK_CRC       1
#define DMX_ONESHOT         2
#define DMX_IMMEDIATE_START 4
#define DMX_KERNEL_CLIENT   0x8000
};
```

struct dmx_pes_filter_params

dmx_pes_filter_params

```
struct dmx_pes_filter_params
{
    __u16          pid;
    dmx_input_t    input;
    dmx_output_t   output;
    dmx_pes_type_t pes_type;
    __u32          flags;
};
```

struct dmx_event

dmx_event

```
struct dmx_event
{
    dmx_event_t          event;
    time_t               timeStamp;
    union
    {
        dmx_scrambling_status_t scrambling;
    } u;
};
```

struct dmx_stc

dmx_stc

```
struct dmx_stc {
    unsigned int num;   /* input : which STC? 0..N */
    unsigned int base;  /* output: divisor for stc to get 90 kHz clock */
    __u64 stc;      /* output: stc in 'base'*90 kHz units */
};
```

struct dmx_caps

dmx_caps

```
 typedef struct dmx_caps {
    __u32 caps;
    int num_decoders;
} dmx_caps_t;
```

enum dmx_source

dmx_source

```
typedef enum dmx_source {
    DMX_SOURCE_FRONT0 = 0,
    DMX_SOURCE_FRONT1,
    DMX_SOURCE_FRONT2,
    DMX_SOURCE_FRONT3,
    DMX_SOURCE_DVR0   = 16,
    DMX_SOURCE_DVR1,
    DMX_SOURCE_DVR2,
    DMX_SOURCE_DVR3
} dmx_source_t;
```

Demux Function Calls

DVB demux open()

Name

DVB demux open()

Synopsis

int **open**(const char **deviceName*, int *flags*)

Arguments

name Name of specific DVB demux device.

flags A bit-wise OR of the following flags:

O_RDONLY	read-only access
O_RDWR	read/write access
O_NONBLOCK	open in non-blocking mode (blocking mode is the default)

Description

This system call, used with a device name of /dev/dvb/adapter0/demux0, allocates a new filter and returns a handle which can be used for subsequent control of that filter. This call has to be made for each filter to be used, i.e. every returned file descriptor is a reference to a single filter. /dev/dvb/adapter0/dvr0 is a logical device to be used for retrieving Transport Streams for digital video recording. When reading from this device a transport stream containing the packets from all PES filters set in the corresponding demux device (/dev/dvb/adapter0/demux0) having the output set to DMX_OUT_TS_TAP. A recorded Transport Stream is replayed by writing to this device.

The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the open() call itself. A device opened in blocking

mode can later be put into non-blocking mode (and vice versa) using the F_SETFL command of the fcntl system call.

Return Value

ENODEV	Device driver not loaded/available.
EINVAL	Invalid argument.
EMFILE	“Too many open files”, i.e. no more filters available.
ENOMEM	The driver failed to allocate enough memory.

DVB demux close()

Name

DVB demux close()

Synopsis

int **close**(int *fd*)

Arguments

fd File descriptor returned by a previous call to *open()*.

Description

This system call deactivates and deallocates a filter that was previously allocated via the open() call.

Return Value

EBADF	fd is not a valid open file descriptor.

DVB demux read()

Name

DVB demux read()

Synopsis

size_t **read**(int *fd*, void **buf*, size_t *count*)

Arguments

fd

File descriptor returned by a previous call to *open()*.

buf Buffer to be filled

count Max number of bytes to read

Description

This system call returns filtered data, which might be section or PES data. The filtered data is transferred from the driver's internal circular buffer to buf. The maximum amount of data to be transferred is implied by count.

Return Value

EWOULDBLOCK	No data to return and O_NONBLOCK was specified.
EBADF	fd is not a valid open file descriptor.
ECRC	Last section had a CRC error - no data returned. The buffer is flushed.
EOVERFLOW	
	The filtered data was not read from the buffer in due time, resulting in non-read data being lost. The buffer is flushed.
ETIMEDOUT	The section was not loaded within the stated timeout period. See ioctl DMX_SET_FILTER for how to set a timeout.
EFAULT	The driver failed to write to the callers buffer due to an invalid *buf pointer.

DVB demux write()

Name

DVB demux write()

Synopsis

ssize_t **write**(int *fd*, const void **buf*, size_t *count*)

Arguments

fd File descriptor returned by a previous call to *open()*.

buf Buffer with data to be written

count Number of bytes at the buffer

Description

This system call is only provided by the logical device /dev/dvb/adapter0/dvr0, associated with the physical demux device that provides the actual DVR functionality. It is used for replay of a digitally recorded Transport Stream. Matching filters have to be defined in the corresponding physical demux device, /dev/dvb/adapter0/demux0. The amount of data to be transferred is implied by count.

Return Value

EWOULDBLOCK	No data was written. This might happen if O_NONBLOCK was specified and there is no more buffer space available (if O_NONBLOCK is not specified the function will block until buffer space is available).
EBUSY	This error code indicates that there are conflicting requests. The corresponding demux device is setup to receive data from the front- end. Make sure that these filters are stopped and that the filters with input set to DMX_IN_DVR are started.
EBADF	fd is not a valid open file descriptor.

DMX_START

Name

DMX_START

Synopsis

int **ioctl**(int *fd*, *DMX_START*)

Arguments

fd File descriptor returned by *open()*.

Description

This ioctl call is used to start the actual filtering operation defined via the ioctl calls DMX_SET_FILTER or DMX_SET_PES_FILTER.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	Invalid argument, i.e. no filtering parameters provided via the DMX_SET_FILTER or DMX_SET_PES_FILTER functions.
EBUSY	This error code indicates that there are conflicting requests. There are active filters filtering data from another input source. Make sure that these filters are stopped before starting this filter.

DMX_STOP

Name

DMX_STOP

Synopsis

int **ioctl**(int *fd*, *DMX_STOP*)

Arguments

fd File descriptor returned by *open()*.

Description

This ioctl call is used to stop the actual filtering operation defined via the ioctl calls DMX_SET_FILTER or DMX_SET_PES_FILTER and started via the DMX_START command.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

DMX_SET_FILTER

Name

DMX_SET_FILTER

Synopsis

int **ioctl**(int *fd*, *DMX_SET_FILTER*, struct *dmx_sct_filter_params* *params)

Arguments

fd File descriptor returned by *open()*.

`params`

Pointer to structure containing filter parameters.

Description

This ioctl call sets up a filter according to the filter and mask parameters provided. A timeout may be defined stating number of seconds to wait for a section to be loaded. A value of 0 means that no timeout should be applied. Finally there is a flag field where it is possible to state whether a section should be CRC-checked, whether the filter should be a ”one-shot” filter, i.e. if the filtering operation should be stopped after the first section is received, and whether the filtering operation should be started immediately (without waiting for a DMX_START ioctl call). If a filter was previously set-up, this filter will be canceled, and the receive buffer will be flushed.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

DMX_SET_PES_FILTER

Name

DMX_SET_PES_FILTER

Synopsis

int **ioctl**(int *fd*, *DMX_SET_PES_FILTER*, struct *dmx_pes_filter_params* *params)

Arguments

fd File descriptor returned by *open()*.

params Pointer to structure containing filter parameters.

Description

This ioctl call sets up a PES filter according to the parameters provided. By a PES filter is meant a filter that is based just on the packet identifier (PID), i.e. no PES header or payload filtering capability is supported.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY	This error code indicates that there are conflicting requests. There are active filters filtering data from another input source. Make sure that these filters are stopped before starting this filter.

DMX_SET_BUFFER_SIZE

Name

DMX_SET_BUFFER_SIZE

Synopsis

int **ioctl**(int *fd*, *DMX_SET_BUFFER_SIZE*, unsigned long *size*)

Arguments

fd File descriptor returned by *open()*.

size Unsigned long size

Description

This ioctl call is used to set the size of the circular buffer used for filtered data. The default size is two maximum sized sections, i.e. if this function is not called a buffer size of 2 * 4096 bytes will be used.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

DMX_GET_EVENT

Name

DMX_GET_EVENT

Synopsis

int **ioctl**(int *fd*, *DMX_GET_EVENT*, struct *dmx_event* *ev)

Arguments

fd File descriptor returned by *open()*.

ev Pointer to the location where the event is to be stored.

Description

This ioctl call returns an event if available. If an event is not available, the behavior depends on whether the device is in blocking or non-blocking mode. In the latter case, the call fails immediately with errno set to EWOULDBLOCK. In the former case, the call blocks until an event becomes available.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EWOULDBLOCK	There is no event pending, and the device is in non-blocking mode.

DMX_GET_STC

Name

DMX_GET_STC

Synopsis

int **ioctl**(int *fd*, *DMX_GET_STC*, struct *dmx_stc* *stc)

Arguments

fd File descriptor returned by *open()*.

stc Pointer to the location where the stc is to be stored.

Description

This ioctl call returns the current value of the system time counter (which is driven by a PES filter of type DMX_PES_PCR). Some hardware supports more than one STC, so you must specify which one by setting the num field of stc before the ioctl (range 0...n). The result is returned in form of a ratio with a 64 bit numerator and a 32 bit denominator, so the real 90kHz STC value is stc->stc / stc->base .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	Invalid stc number.

DMX_GET_PES_PIDS

Name

DMX_GET_PES_PIDS

Synopsis

int **ioctl**(fd, *DMX_GET_PES_PIDS*, __u16 *pids[5]*)

Arguments

fd File descriptor returned by *open()*.

pids Undocumented.

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

DMX_GET_CAPS

Name

DMX_GET_CAPS

Synopsis

int **ioctl**(fd, *DMX_GET_CAPS*, struct *dmx_caps* *caps*)

Arguments

fd File descriptor returned by *open()*.

caps Pointer to struct *dmx_caps*

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

DMX_SET_SOURCE

Name

DMX_SET_SOURCE

Synopsis

int **ioctl**(fd, *DMX_SET_SOURCE*, struct *dmx_source* *src*)

Arguments

fd File descriptor returned by *open()*.

src Undocumented.

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

DMX_ADD_PID

Name

DMX_ADD_PID

Synopsis

int **ioctl**(fd, *DMX_ADD_PID*, __u16 *pid)

Arguments

fd File descriptor returned by *open()*.

pid PID number to be filtered.

Description

This ioctl call allows to add multiple PIDs to a transport stream filter previously set up with DMX_SET_PES_FILTER and output equal to DMX_OUT_TSDEMUX_TAP.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

DMX_REMOVE_PID

Name

DMX_REMOVE_PID

Synopsis

int **ioctl**(fd, *DMX_REMOVE_PID*, __u16 *pid)

Arguments

fd File descriptor returned by *open()*.

pid PID of the PES filter to be removed.

Description

This ioctl call allows to remove a PID when multiple PIDs are set on a transport stream filter, e. g. a filter previously set up with output equal to DMX_OUT_TSDEMUX_TAP, created via either DMX_SET_PES_FILTER or DMX_ADD_PID.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

1.3.4 DVB CA Device

The DVB CA device controls the conditional access hardware. It can be accessed through `/dev/dvb/adapter?/ca?`. Data types and and ioctl definitions can be accessed by including `linux/dvb/ca.h` in your application.

CA Data Types

ca_slot_info

ca_slot_info_t

```
typedef struct ca_slot_info {
    int num;                /* slot number */

    int type;               /* CA interface this slot supports */
#define CA_CI            1     /* CI high level interface */
#define CA_CI_LINK       2     /* CI link layer level interface */
#define CA_CI_PHYS       4     /* CI physical layer level interface */
#define CA_DESCR         8     /* built-in descrambler */
#define CA_SC          128     /* simple smart card interface */

    unsigned int flags;
#define CA_CI_MODULE_PRESENT 1 /* module (or card) inserted */
#define CA_CI_MODULE_READY   2
} ca_slot_info_t;
```

ca_descr_info

ca_descr_info_t

```
typedef struct ca_descr_info {
    unsigned int num;  /* number of available descramblers (keys) */
    unsigned int type; /* type of supported scrambling system */
#define CA_ECD           1
#define CA_NDS           2
#define CA_DSS           4
} ca_descr_info_t;
```

ca_caps

ca_caps_t

```
typedef struct ca_caps {
    unsigned int slot_num;  /* total number of CA card and module slots */
    unsigned int slot_type; /* OR of all supported types */
    unsigned int descr_num; /* total number of descrambler slots (keys) */
    unsigned int descr_type;/* OR of all supported types */
 } ca_cap_t;
```

ca_msg

ca_msg_t

```
/* a message to/from a CI-CAM */
typedef struct ca_msg {
    unsigned int index;
    unsigned int type;
    unsigned int length;
    unsigned char msg[256];
} ca_msg_t;
```

ca_descr

ca_descr_t

```
typedef struct ca_descr {
    unsigned int index;
    unsigned int parity;
    unsigned char cw[8];
} ca_descr_t;
```

ca_pid

ca-pid

```
typedef struct ca_pid {
    unsigned int pid;
    int index;      /* -1 == disable*/
} ca_pid_t;
```

CA Function Calls

DVB CA open()

Name

DVB CA open()

Synopsis

int **open**(const char *_name_, int _flags_)

Arguments

name Name of specific DVB CA device.

flags A bit-wise OR of the following flags:

O_RDONLY	read-only access
O_RDWR	read/write access
O_NONBLOCK	open in non-blocking mode (blocking mode is the default)

Description

This system call opens a named ca device (e.g. /dev/ost/ca) for subsequent use.

When an open() call has succeeded, the device will be ready for use. The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the open() call itself. A device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the F_SETFL command of the fcntl system call. This is a standard system call, documented in the Linux manual page for fcntl. Only one user can open the CA Device in O_RDWR mode. All other attempts to open the device in this mode will fail, and an error code will be returned.

Return Value

ENODEV	Device driver not loaded/available.
EINTERNAL	Internal error.
EBUSY	Device or resource busy.
EINVAL	Invalid argument.

DVB CA close()

Name

DVB CA close()

Synopsis

int **close**(int *fd*)

Arguments

fd File descriptor returned by a previous call to *open()*.

Description

This system call closes a previously opened CA device.

Return Value

EBADF	fd is not a valid open file descriptor.

CA_RESET

Name

CA_RESET

Synopsis

int **ioctl**(fd, *CA_RESET*)

Arguments

fd File descriptor returned by a previous call to *open()*.

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_CAP

Name

CA_GET_CAP

Synopsis

int **ioctl**(fd, *CA_GET_CAP*, struct *ca_caps* **caps*)

Arguments

fd File descriptor returned by a previous call to *open()*.

caps Pointer to struct *ca_caps*.

struct **ca_caps**

Table 1.189: struct ca_caps

type	name	description
unsigned int	slot_num	total number of CA card and module slots
unsigned int	slot_type	bitmask with all supported slot types
unsigned int	descr_num	total number of descrambler slots (keys)
unsigned int	descr_type	bit mask with all supported descr types

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_SLOT_INFO

Name

CA_GET_SLOT_INFO

Synopsis

int **ioctl**(fd, *CA_GET_SLOT_INFO*, struct *ca_slot_info* *_info_)

Arguments

fd File descriptor returned by a previous call to *open()*.

info Pointer to struct c:type:*ca_slot_info*.

Table 1.190: ca_slot_info types

type	name	description
CA_CI	1	CI high level interface
CA_CI_LINK	2	CI link layer level interface
CA_CI_PHYS	4	CI physical layer level interface
CA_DESCR	8	built-in descrambler
CA_SC	128	simple smart card interface

Table 1.191: ca_slot_info flags

type	name	description
CA_CI_MODULE_PRESENT	1	module (or card) inserted
CA_CI_MODULE_READY	2	

ca_slot_info

Table 1.192: struct ca_slot_info

type	name	description
int	num	slot number
int	type	CA interface this slot supports, as defined at *ca_slot_info types* .
unsigned int	flags	flags as defined at *ca_slot_info flags* .

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_DESCR_INFO

Name

CA_GET_DESCR_INFO

Synopsis

int **ioctl**(fd, *CA_GET_DESCR_INFO*, struct *ca_descr_info* *desc)

Arguments

fd File descriptor returned by a previous call to *open()*.

desc Pointer to struct *ca_descr_info*.

struct **ca_descr_info**

Table 1.193: struct ca_descr_info

type	name	description
unsigned int	num	number of available descramblers (keys)
unsigned int	type	type of supported scrambling system. Valid values are: `CA_ECD`, `CA_NDS` and `CA_DSS`.

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_MSG

Name

CA_GET_MSG

Synopsis

int **ioctl**(fd, *CA_GET_MSG*, struct *ca_msg* *msg)

Arguments

fd File descriptor returned by a previous call to *open()*.

msg Pointer to struct *ca_msg*.

struct **ca_msg**

Table 1.194: struct ca_msg

type	name	description
unsigned int	index	
unsigned int	type	
unsigned int	length	
unsigned char	msg[256]	

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

CA_SEND_MSG

Name

CA_SEND_MSG

Synopsis

int **ioctl**(fd, *CA_SEND_MSG*, struct *ca_msg* *msg)

Arguments

fd File descriptor returned by a previous call to *open()*.

msg Pointer to struct *ca_msg*.

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

CA_SET_DESCR

Name

CA_SET_DESCR

Synopsis

int **ioctl**(fd, *CA_SET_DESCR*, struct *ca_descr* *desc)

Arguments

fd File descriptor returned by a previous call to *open()*.

msg Pointer to struct *ca_descr*.

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

CA_SET_PID

Name

CA_SET_PID

Synopsis

int **ioctl**(fd, *CA_SET_PID*, struct *ca_pid* *pid)

Arguments

fd File descriptor returned by a previous call to *open()*.

pid Pointer to struct *ca_pid*.

ca_pid

Table 1.195: struct ca_pid

unsigned int	pid	Program ID
int	index	PID index. Use -1 to disable.

Description

Note:

This ioctl is undocumented. Documentation is welcome.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

1.3.5 DVB Network API

The DVB net device controls the mapping of data packages that are part of a transport stream to be mapped into a virtual network interface, visible through the standard Linux network protocol stack.

Currently, two encapsulations are supported:

- Multi Protocol Encapsulation (MPE)
- Ultra Lightweight Encapsulation (ULE)

In order to create the Linux virtual network interfaces, an application needs to tell to the Kernel what are the PIDs and the encapsulation types that are present on the transport stream. This is done through `/dev/dvb/adapter?/net?` device node. The data will be available via virtual dvb?_? network interfaces, and will be controlled/routed via the standard ip tools (like ip, route, netstat, ifconfig, etc).

Data types and and ioctl definitions are defined via `linux/dvb/net.h` header.

1.3.6 DVB net Function Calls

ioctl NET_ADD_IF

Name

NET_ADD_IF - Creates a new network interface for a given Packet ID.

Synopsis

int **ioctl**(int *fd*, *NET_ADD_IF*, struct *dvb_net_if* **net_if*)

Arguments

fd File descriptor returned by *open()* .

net_if pointer to struct *dvb_net_if*

Description

The NET_ADD_IF ioctl system call selects the Packet ID (PID) that contains a TCP/IP traffic, the type of encapsulation to be used (MPE or ULE) and the interface number for the new interface to be created. When the system call successfully returns, a new virtual network interface is created.

The struct *dvb_net_if*::ifnum field will be filled with the number of the created interface.

dvb_net_if

Table 1.196: struct dvb_net_if

ID	Description
pid	Packet ID (PID) of the MPEG-TS that contains data
ifnum	number of the DVB interface.
feed-type	Encapsulation type of the feed. It can be: DVB_NET_FEEDTYPE_MPE for MPE encoding or DVB_NET_FEEDTYPE_ULE for ULE encoding.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl NET_REMOVE_IF

Name

NET_REMOVE_IF - Removes a network interface.

Synopsis

int **ioctl**(int *fd*, *NET_REMOVE_IF*, int *ifnum*)

Arguments

fd File descriptor returned by *open()* .

net_if number of the interface to be removed

Description

The NET_REMOVE_IF ioctl deletes an interface previously created via *NET_ADD_IF* .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl NET_GET_IF

Name

NET_GET_IF - Read the configuration data of an interface created via - *NET_ADD_IF* .

Synopsis

int **ioctl**(int *fd*, *NET_GET_IF*, struct *dvb_net_if* **net_if*)

Arguments

fd File descriptor returned by *open()* .

net_if pointer to struct *dvb_net_if*

Description

The NET_GET_IF ioctl uses the interface number given by the struct *dvb_net_if*::ifnum field and fills the content of struct *dvb_net_if* with the packet ID and encapsulation type used on such interface. If the interface was not created yet with *NET_ADD_IF* , it will return -1 and fill the `errno` with `EINVAL` error code.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

1.3.7 DVB Deprecated APIs

The APIs described here are kept only for historical reasons. There's just one driver for a very legacy hardware that uses this API. No modern drivers should use it. Instead, audio and video should be using the V4L2 and ALSA APIs, and the pipelines should be set using the Media Controller API

DVB Video Device

The DVB video device controls the MPEG2 video decoder of the DVB hardware. It can be accessed through **/dev/dvb/adapter0/video0**. Data types and and ioctl definitions can be accessed by including **linux/dvb/video.h** in your application.

Note that the DVB video device only controls decoding of the MPEG video stream, not its presentation on the TV or computer screen. On PCs this is typically handled by an associated video4linux device, e.g. **/dev/video**, which allows scaling and defining output windows.

Some DVB cards don't have their own MPEG decoder, which results in the omission of the audio and video device as well as the video4linux device.

The ioctls that deal with SPUs (sub picture units) and navigation packets are only supported on some MPEG decoders made for DVD playback.

These ioctls were also used by V4L2 to control MPEG decoders implemented in V4L2. The use of these ioctls for that purpose has been made obsolete and proper V4L2 ioctls or controls have been created to replace that functionality.

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;
```

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;
```

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
    __kernel_time_t timestamp;
    union {
        video_size_t size;
        unsigned int frame_rate;    /* in frames per 1000sec */
        unsigned char vsync_field;  /* unknown/odd/even/progressive */
    } u;
};
```

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 seleted 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.

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
```

video_system_t

A call to VIDEO_SET_SYSTEM sets the desired video system for TV output. The following system types can be set:

```
typedef enum {
     VIDEO_SYSTEM_PAL,
     VIDEO_SYSTEM_NTSC,
     VIDEO_SYSTEM_PALN,
     VIDEO_SYSTEM_PALNc,
     VIDEO_SYSTEM_PALM,
     VIDEO_SYSTEM_NTSC60,
     VIDEO_SYSTEM_PAL60,
     VIDEO_SYSTEM_PALM60
} video_system_t;
```

video_highlight

struct video_highlight

Calling the ioctl VIDEO_SET_HIGHLIGHTS posts the SPU highlight information. The call expects the following format for that information:

```
typedef
struct video_highlight {
    boolean active;      /*    1=show highlight, 0=hide highlight */
    uint8_t contrast1;   /*    7- 4  Pattern pixel contrast */
                 /*    3- 0  Background pixel contrast */
    uint8_t contrast2;   /*    7- 4  Emphasis pixel-2 contrast */
                 /*    3- 0  Emphasis pixel-1 contrast */
    uint8_t color1;      /*    7- 4  Pattern pixel color */
                 /*    3- 0  Background pixel color */
    uint8_t color2;      /*    7- 4  Emphasis pixel-2 color */
                 /*    3- 0  Emphasis pixel-1 color */
    uint32_t ypos;       /*   23-22  auto action mode */
                 /*   21-12  start y */
                 /*    9- 0  end y */
    uint32_t xpos;       /*   23-22  button color number */
                 /*   21-12  start x */
                 /*    9- 0  end x */
} video_highlight_t;
```

video_spu

struct video_spu

Calling VIDEO_SET_SPU deactivates or activates SPU decoding, according to the following format:

```
typedef
struct video_spu {
    boolean active;
    int stream_id;
} video_spu_t;
```

video_spu_palette

struct video_spu_palette

The following structure is used to set the SPU palette by calling VIDEO_SPU_PALETTE:

```
typedef
struct video_spu_palette {
    int length;
    uint8_t *palette;
} video_spu_palette_t;
```

video_navi_pack

struct video_navi_pack

In order to get the navigational data the following structure has to be passed to the ioctl VIDEO_GET_NAVI:

```
typedef
struct video_navi_pack {
    int length;         /* 0 ... 1024 */
```

```
    uint8_t data[1024];
} video_navi_pack_t;
```

video_attributes_t

The following attributes can be set by a call to VIDEO_SET_ATTRIBUTES:

```
typedef uint16_t video_attributes_t;
/*   bits: descr. */
/*   15-14 Video compression mode (0=MPEG-1, 1=MPEG-2) */
/*   13-12 TV system (0=525/60, 1=625/50) */
/*   11-10 Aspect ratio (0=4:3, 3=16:9) */
/*    9- 8 permitted display mode on 4:3 monitor (0=both, 1=only pan-sca */
/*    7    line 21-1 data present in GOP (1=yes, 0=no) */
/*    6    line 21-2 data present in GOP (1=yes, 0=no) */
/*    5- 3 source resolution (0=720x480/576, 1=704x480/576, 2=352x480/57 */
/*    2    source letterboxed (1=yes, 0=no) */
/*    0    film/camera mode (0=camera, 1=film (625/50 only)) */
```

Video Function Calls

dvb video open()

Name

dvb video open()

Attention:

This ioctl is deprecated.

Synopsis

int **open**(const char **deviceName*, int *flags*)

Arguments

const char *deviceName	Name of specific video device.
int flags	A bit-wise OR of the following flags:
	O_RDONLY read-only access
	O_RDWR read/write access
	O_NONBLOCK open in non-blocking mode
	(blocking mode is the default)

Description

This system call opens a named video device (e.g. /dev/dvb/adapter0/video0) for subsequent use.

When an open() call has succeeded, the device will be ready for use. The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the open() call itself. A device opened in blocking mode can later be put

into non-blocking mode (and vice versa) using the F_SETFL command of the fcntl system call. This is a standard system call, documented in the Linux manual page for fcntl. Only one user can open the Video Device in O_RDWR mode. All other attempts to open the device in this mode will fail, and an error-code will be returned. If the Video Device is opened in O_RDONLY mode, the only ioctl call that can be used is VIDEO_GET_STATUS. All other call will return an error code.

Return Value

ENODEV	Device driver not loaded/available.
EINTERNAL	Internal error.
EBUSY	Device or resource busy.
EINVAL	Invalid argument.

dvb video close()

Name

dvb video close()

Attention:

This ioctl is deprecated.

Synopsis

int **close**(int *fd*)

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This system call closes a previously opened video device.

Return Value

EBADF	fd is not a valid open file descriptor.

dvb video write()

Name

dvb video write()

Attention:

This ioctl is deprecated.

Synopsis

size_t **write**(int *fd*, const void **buf*, size_t *count*)

Arguments

int fd	File descriptor returned by a previous call to open().
void *buf	Pointer to the buffer containing the PES data.
size_t count	Size of buf.

Description

This system call can only be used if VIDEO_SOURCE_MEMORY is selected in the ioctl call VIDEO_SELECT_SOURCE. The data provided shall be in PES format, unless the capability allows other formats. If O_NONBLOCK is not specified the function will block until buffer space is available. The amount of data to be transferred is implied by count.

Return Value

EPERM	Mode VIDEO_SOURCE_MEMORY not selected.
ENOMEM	Attempted to write more data than the internal buffer can hold.
EBADF	fd is not a valid open file descriptor.

VIDEO_STOP

Name

VIDEO_STOP

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_STOP*, boolean *mode*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_STOP for this command.
Boolean mode	Indicates how the screen shall be handled.
	TRUE: Blank screen when stop.
	FALSE: Show last decoded frame.

Description

This ioctl is for DVB devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* instead.

This ioctl call asks the Video Device to stop playing the current stream. Depending on the input parameter, the screen can be blanked out or displaying the last decoded frame.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_PLAY

Name

VIDEO_PLAY

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_PLAY*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_PLAY for this command.

Description

This ioctl is for DVB devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* instead.

This ioctl call asks the Video Device to start playing a video stream from the selected source.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_FREEZE

Name

VIDEO_FREEZE

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_FREEZE*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_FREEZE for this command.

Description

This ioctl is for DVB devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* instead.

This ioctl call suspends the live video stream being played. Decoding and playing are frozen. It is then possible to restart the decoding and playing process of the video stream using the VIDEO_CONTINUE command. If VIDEO_SOURCE_MEMORY is selected in the ioctl call VIDEO_SELECT_SOURCE, the DVB subsystem will not decode any more data until the ioctl call VIDEO_CONTINUE or VIDEO_PLAY is performed.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_CONTINUE

Name

VIDEO_CONTINUE

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_CONTINUE*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_CONTINUE for this command.

Description

This ioctl is for DVB devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* instead.

This ioctl call restarts decoding and playing processes of the video stream which was played before a call to VIDEO_FREEZE was made.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SELECT_SOURCE

Name

VIDEO_SELECT_SOURCE

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SELECT_SOURCE*, *video_stream_source_t source*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SELECT_SOURCE for this command.
video_stream_source_t source	Indicates which source shall be used for the Video stream.

Description

This ioctl is for DVB devices only. This ioctl was also supported by the V4L2 ivtv driver, but that has been replaced by the ivtv-specific `IVTV_IOC_PASSTHROUGH_MODE` ioctl.

This ioctl call informs the video device which source shall be used for the input data. The possible sources are demux or memory. If memory is selected, the data is fed to the video device through the write command.

video_stream_source_t

```
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;
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_BLANK

Name

VIDEO_SET_BLANK

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SET_BLANK*, boolean *mode*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_BLANK for this command.
boolean mode	TRUE: Blank screen when stop.
	FALSE: Show last decoded frame.

Description

This ioctl call asks the Video Device to blank out the picture.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_STATUS

Name

VIDEO_GET_STATUS

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_GET_STATUS*, struct *video_status* **status*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_STATUS for this command.
struct video_status *status	Returns the current status of the Video Device.

Description

This ioctl call asks the Video Device to return the current status of the device.

video_status

```
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 */
};
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_FRAME_COUNT

Name

VIDEO_GET_FRAME_COUNT

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *VIDEO_GET_FRAME_COUNT*, __u64 **pts*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_FRAME_COUNT for this command.
__u64 *pts	Returns the number of frames displayed since the decoder was started.

Description

This ioctl is obsolete. Do not use in new drivers. For V4L2 decoders this ioctl has been replaced by the `V4L2_CID_MPEG_VIDEO_DEC_FRAME` control.

This ioctl call asks the Video Device to return the number of displayed frames since the decoder was started.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_PTS

Name

VIDEO_GET_PTS

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *VIDEO_GET_PTS*, __u64 **pts*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_PTS for this command.
__u64 *pts	Returns the 33-bit timestamp as defined in ITU T-REC-H.222.0 / ISO/IEC 13818-1. The PTS should belong to the currently played frame if possible, but may also be a value close to it like the PTS of the last decoded frame or the last PTS extracted by the PES parser.

Description

This ioctl is obsolete. Do not use in new drivers. For V4L2 decoders this ioctl has been replaced by the `V4L2_CID_MPEG_VIDEO_DEC_PTS` control.

This ioctl call asks the Video Device to return the current PTS timestamp.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_FRAME_RATE

Name

VIDEO_GET_FRAME_RATE

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *VIDEO_GET_FRAME_RATE*, unsigned int **rate*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_FRAME_RATE for this command.
unsigned int *rate	Returns the framerate in number of frames per 1000 seconds.

Description

This ioctl call asks the Video Device to return the current framerate.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_EVENT

Name

VIDEO_GET_EVENT

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_GET_EVENT*, struct *video_event* **ev*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_EVENT for this command.
struct video_event *ev	Points to the location where the event, if any, is to be stored.

Description

This ioctl is for DVB devices only. To get events from a V4L2 decoder use the V4L2 *ioctl VIDIOC_DQEVENT* ioctl instead.

This ioctl call returns an event of type video_event if available. If an event is not available, the behavior depends on whether the device is in blocking or non-blocking mode. In the latter case, the call fails immediately with errno set to EWOULDBLOCK. In the former case, the call blocks until an event becomes available. The standard Linux poll() and/or select() system calls can be used with the device file descriptor to watch for new events. For select(), the file descriptor should be included in the exceptfds argument, and for poll(), POLLPRI should be specified as the wake-up condition. Read-only permissions are sufficient for this ioctl call.

video_event

```
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
        __kernel_time_t timestamp;
        union {
                video_size_t size;
                unsigned int frame_rate;        /* in frames per 1000sec */
                unsigned char vsync_field;      /* unknown/odd/even/progressive */
        } u;
};
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EWOULDBLOCK	There is no event pending, and the device is in non-blocking mode.
EOVERFLOW	Overflow in event queue - one or more events were lost.

VIDEO_COMMAND

Name

VIDEO_COMMAND

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *VIDEO_COMMAND*, struct *video_command *cmd*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_COMMAND for this command.
struct video_command *cmd	Commands the decoder.

Description

This ioctl is obsolete. Do not use in new drivers. For V4L2 decoders this ioctl has been replaced by the *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* ioctl.

This ioctl commands the decoder. The `video_command` struct is a subset of the `v4l2_decoder_cmd` struct, so refer to the *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* documentation for more information.

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;
        };
};
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_TRY_COMMAND

Name

VIDEO_TRY_COMMAND

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *VIDEO_TRY_COMMAND*, struct *video_command *cmd*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_TRY_COMMAND for this command.
struct video_command *cmd	Try a decoder command.

Description

This ioctl is obsolete. Do not use in new drivers. For V4L2 decoders this ioctl has been replaced by the *VIDIOC_TRY_DECODER_CMD* ioctl.

This ioctl tries a decoder command. The `video_command` struct is a subset of the `v4l2_decoder_cmd` struct, so refer to the *VIDIOC_TRY_DECODER_CMD* documentation for more information.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_SIZE

Name

VIDEO_GET_SIZE

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *VIDEO_GET_SIZE*, *video_size_t *size*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_SIZE for this command.
video_size_t *size	Returns the size and aspect ratio.

Description

This ioctl returns the size and aspect ratio.

video_size_t

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_DISPLAY_FORMAT

Name

VIDEO_SET_DISPLAY_FORMAT

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SET_DISPLAY_FORMAT*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_DISPLAY_FORMAT for this command.
video_display_format_t format	Selects the video format to be used.

Description

This ioctl call asks the Video Device to select the video format to be applied by the MPEG chip on the video.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_STILLPICTURE

Name

VIDEO_STILLPICTURE

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_STILLPICTURE*, struct *video_still_picture* *sp)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_STILLPICTURE for this command.
struct video_still_picture *sp	Pointer to a location where an I-frame and size is stored.

Description

This ioctl call asks the Video Device to display a still picture (I-frame). The input data shall contain an I-frame. If the pointer is NULL, then the current displayed still picture is blanked.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_FAST_FORWARD

Name

VIDEO_FAST_FORWARD

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_FAST_FORWARD*, int *nFrames*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_FAST_FORWARD for this command.
int nFrames	The number of frames to skip.

Description

This ioctl call asks the Video Device to skip decoding of N number of I-frames. This call can only be used if VIDEO_SOURCE_MEMORY is selected.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EPERM	Mode VIDEO_SOURCE_MEMORY not selected.

VIDEO_SLOWMOTION

Name

VIDEO_SLOWMOTION

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SLOWMOTION*, int *nFrames*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SLOWMOTION for this command.
int nFrames	The number of times to repeat each frame.

Description

This ioctl call asks the video device to repeat decoding frames N number of times. This call can only be used if VIDEO_SOURCE_MEMORY is selected.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EPERM	Mode VIDEO_SOURCE_MEMORY not selected.

VIDEO_GET_CAPABILITIES

Name

VIDEO_GET_CAPABILITIES

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_GET_CAPABILITIES*, unsigned int *cap)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_CAPABILITIES for this command.
unsigned int *cap	Pointer to a location where to store the capability information.

Description

This ioctl call asks the video device about its decoding capabilities. On success it returns and integer which has bits set according to the defines in section ??.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_ID

Name

VIDEO_SET_ID

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(int *fd*, *VIDEO_SET_ID*, int *id*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_ID for this command.
int id	video sub-stream id

Description

This ioctl selects which sub-stream is to be decoded if a program or system stream is sent to the video device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	Invalid sub-stream id.

VIDEO_CLEAR_BUFFER

Name

VIDEO_CLEAR_BUFFER

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_CLEAR_BUFFER*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_CLEAR_BUFFER for this command.

Description

This ioctl call clears all video buffers in the driver and in the decoder hardware.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_STREAMTYPE

Name

VIDEO_SET_STREAMTYPE

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SET_STREAMTYPE*, int *type*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_STREAMTYPE for this command.
int type	stream type

Description

This ioctl tells the driver which kind of stream to expect being written to it. If this call is not used the default of video PES is used. Some drivers might not support this call and always expect PES.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_FORMAT

Name

VIDEO_SET_FORMAT

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SET_FORMAT*, *video_format_t format*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_FORMAT for this command.
video_format_t format	video format of TV as defined in section ??.

Description

This ioctl sets the screen format (aspect ratio) of the connected output device (TV) so that the output of the decoder can be adjusted accordingly.

video_format_t

```
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;
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	format is not a valid video format.

VIDEO_SET_SYSTEM

Name

VIDEO_SET_SYSTEM

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SET_SYSTEM*, video_system_t *system*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_FORMAT for this command.
video_system_t system	video system of TV output.

Description

This ioctl sets the television output format. The format (see section ??) may vary from the color format of the displayed MPEG stream. If the hardware is not able to display the requested format the call will return an error.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	system is not a valid or supported video system.

VIDEO_SET_HIGHLIGHT

Name

VIDEO_SET_HIGHLIGHT

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SET_HIGHLIGHT*, struct *video_highlight *vhilite*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_HIGHLIGHT for this command.
video_highlight_t *vhilite	SPU Highlight information according to section ??.

Description

This ioctl sets the SPU highlight information for the menu access of a DVD.

video_highlight

```
typedef
struct video_highlight {
        int     active;      /*    1=show highlight, 0=hide highlight */
        __u8    contrast1;   /*    7- 4  Pattern pixel contrast */
                             /*    3- 0  Background pixel contrast */
        __u8    contrast2;   /*    7- 4  Emphasis pixel-2 contrast */
                             /*    3- 0  Emphasis pixel-1 contrast */
        __u8    color1;      /*    7- 4  Pattern pixel color */
                             /*    3- 0  Background pixel color */
        __u8    color2;      /*    7- 4  Emphasis pixel-2 color */
                             /*    3- 0  Emphasis pixel-1 color */
        __u32    ypos;       /*   23-22  auto action mode */
                             /*   21-12  start y */
                             /*    9- 0  end y */
        __u32    xpos;       /*   23-22  button color number */
                             /*   21-12  start x */
                             /*    9- 0  end x */
} video_highlight_t;
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_SPU

Name

VIDEO_SET_SPU

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SET_SPU*, struct *video_spu* *spu)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_SPU for this command.
video_spu_t *spu	SPU decoding (de)activation and subid setting according to section ??.

Description

This ioctl activates or deactivates SPU decoding in a DVD input stream. It can only be used, if the driver is able to handle a DVD stream.

struct **video_spu**

```
typedef struct video_spu {
        int active;
        int stream_id;
} video_spu_t;
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	input is not a valid spu setting or driver cannot handle SPU.

VIDEO_SET_SPU_PALETTE

Name

VIDEO_SET_SPU_PALETTE

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SET_SPU_PALETTE*, struct *video_spu_palette* *palette)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_SPU_PALETTE for this command.
video_spu_palette_t *palette	SPU palette according to section ??.

Description

This ioctl sets the SPU color palette.

video_spu_palette

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	input is not a valid palette or driver doesn't handle SPU.

VIDEO_GET_NAVI

Name

VIDEO_GET_NAVI

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_GET_NAVI*, struct *video_navi_pack* *navipack)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_NAVI for this command.
video_navi_pack_t *navipack	PCI or DSI pack (private stream 2) according to section ??.

Description

This ioctl returns navigational information from the DVD stream. This is especially needed if an encoded stream has to be decoded by the hardware.

video_navi_pack

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EFAULT	driver is not able to return navigational information

VIDEO_SET_ATTRIBUTES

Name

VIDEO_SET_ATTRIBUTES

Attention:

This ioctl is deprecated.

Synopsis

int **ioctl**(fd, *VIDEO_SET_ATTRIBUTE*, *video_attributes_t vattr*)

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_ATTRIBUTE for this command.
video_attributes_t vattr	video attributes according to section ??.

Description

This ioctl is intended for DVD playback and allows you to set certain information about the stream. Some hardware may not need this information, but the call also tells the hardware to prepare for DVD playback.

video_attributes_t

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	input is not a valid attribute setting.

DVB Audio Device

The DVB audio device controls the MPEG2 audio decoder of the DVB hardware. It can be accessed through `/dev/dvb/adapter?/audio?`. Data types and and ioctl definitions can be accessed by including `linux/dvb/audio.h` in your application.

Please note that some DVB cards don't have their own MPEG decoder, which results in the omission of the audio and video device.

These ioctls were also used by V4L2 to control MPEG decoders implemented in V4L2. The use of these ioctls for that purpose has been made obsolete and proper V4L2 ioctls or controls have been created to replace that functionality.

Audio Data Types

This section describes the structures, data types and defines used when talking to the audio device.

audio_stream_source

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

```
typedef enum {
    AUDIO_SOURCE_DEMUX,
    AUDIO_SOURCE_MEMORY
} audio_stream_source_t;
```

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

audio_play_state

The following values can be returned by the AUDIO_GET_STATUS call representing the state of audio playback.

```
typedef enum {
    AUDIO_STOPPED,
    AUDIO_PLAYING,
    AUDIO_PAUSED
} audio_play_state_t;
```

audio_channel_select

The audio channel selected via AUDIO_CHANNEL_SELECT is determined by the following values.

```
typedef enum {
    AUDIO_STEREO,
    AUDIO_MONO_LEFT,
    AUDIO_MONO_RIGHT,
    AUDIO_MONO,
    AUDIO_STEREO_SWAPPED
} audio_channel_select_t;
```

audio_status

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

```
typedef struct audio_status {
    boolean AV_sync_state;
    boolean mute_state;
    audio_play_state_t play_state;
```

```
    audio_stream_source_t stream_source;
    audio_channel_select_t channel_select;
    boolean bypass_mode;
    audio_mixer_t mixer_state;
} audio_status_t;
```

audio_mixer

The following structure is used by the AUDIO_SET_MIXER call to set the audio volume.

```
typedef struct audio_mixer {
    unsigned int volume_left;
    unsigned int volume_right;
} audio_mixer_t;
```

audio encodings

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

```
#define AUDIO_CAP_DTS    1
#define AUDIO_CAP_LPCM   2
#define AUDIO_CAP_MP1    4
#define AUDIO_CAP_MP2    8
#define AUDIO_CAP_MP3   16
#define AUDIO_CAP_AAC   32
#define AUDIO_CAP_OGG   64
#define AUDIO_CAP_SDDS 128
#define AUDIO_CAP_AC3  256
```

audio_karaoke

The ioctl AUDIO_SET_KARAOKE uses the following format:

```
typedef
struct audio_karaoke {
    int vocal1;
    int vocal2;
    int melody;
} audio_karaoke_t;
```

If Vocal1 or Vocal2 are non-zero, they get mixed into left and right t at 70% each. If both, Vocal1 and Vocal2 are non-zero, Vocal1 gets mixed into the left channel and Vocal2 into the right channel at 100% each. Ff Melody is non-zero, the melody channel gets mixed into left and right.

audio_attributes

The following attributes can be set by a call to AUDIO_SET_ATTRIBUTES:

```
typedef uint16_t audio_attributes_t;
/*   bits: descr. */
/*   15-13 audio coding mode (0=ac3, 2=mpeg1, 3=mpeg2ext, 4=LPCM, 6=DTS, */
/*   12    multichannel extension */
/*   11-10 audio type (0=not spec, 1=language included) */
/*    9- 8 audio application mode (0=not spec, 1=karaoke, 2=surround) */
/*    7- 6 Quantization / DRC (mpeg audio: 1=DRC exists)(lpcm: 0=16bit,  */
/*    5- 4 Sample frequency fs (0=48kHz, 1=96kHz) */
/*    2- 0 number of audio channels (n+1 channels) */
```

Audio Function Calls

DVB audio open()

Name

DVB audio open()

Attention:

This ioctl is deprecated

Synopsis

int **open**(const char *_deviceName_, int _flags_)

Arguments

const char *deviceName	Name of specific audio device.
int flags	A bit-wise OR of the following flags:
	O_RDONLY read-only access
	O_RDWR read/write access
	O_NONBLOCK open in non-blocking mode
	(blocking mode is the default)

Description

This system call opens a named audio device (e.g. /dev/dvb/adapter0/audio0) for subsequent use. When an open() call has succeeded, the device will be ready for use. The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the open() call itself. A device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the F_SETFL command of the fcntl system call. This is a standard system call, documented in the Linux manual page for fcntl. Only one user can open the Audio Device in O_RDWR mode. All other attempts to open the device in this mode will fail, and an error code will be returned. If the Audio Device is opened in O_RDONLY mode, the only ioctl call that can be used is AUDIO_GET_STATUS. All other call will return with an error code.

Return Value

ENODEV	Device driver not loaded/available.
EBUSY	Device or resource busy.
EINVAL	Invalid argument.

DVB audio close()

Name

DVB audio close()

Attention:

This ioctl is deprecated

Synopsis

int **close**(int *fd*)

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This system call closes a previously opened audio device.

Return Value

EBADF	fd is not a valid open file descriptor.

DVB audio write()

Name

DVB audio write()

Attention:

This ioctl is deprecated

Synopsis

size_t **write**(int *fd*, const void **buf*, size_t *count*)

Arguments

int fd	File descriptor returned by a previous call to open().
void *buf	Pointer to the buffer containing the PES data.
size_t count	Size of buf.

Description

This system call can only be used if AUDIO_SOURCE_MEMORY is selected in the ioctl call AUDIO_SELECT_SOURCE. The data provided shall be in PES format. If O_NONBLOCK is not specified the function will block until buffer space is available. The amount of data to be transferred is implied by count.

Return Value

EPERM	Mode AUDIO_SOURCE_MEMORY not selected.
ENOMEM	Attempted to write more data than the internal buffer can hold.
EBADF	fd is not a valid open file descriptor.

AUDIO_STOP

Name

AUDIO_STOP

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_STOP*)

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This ioctl call asks the Audio Device to stop playing the current stream.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_PLAY

Name

AUDIO_PLAY

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_PLAY*)

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This ioctl call asks the Audio Device to start playing an audio stream from the selected source.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_PAUSE

Name

AUDIO_PAUSE

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_PAUSE*)

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This ioctl call suspends the audio stream being played. Decoding and playing are paused. It is then possible to restart again decoding and playing process of the audio stream using AUDIO_CONTINUE command.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_CONTINUE

Name

AUDIO_CONTINUE

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_CONTINUE*)

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This ioctl restarts the decoding and playing process previously paused with AUDIO_PAUSE command.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SELECT_SOURCE

Name

AUDIO_SELECT_SOURCE

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_SELECT_SOURCE*, struct *audio_stream_source* *source)

Arguments

int fd	File descriptor returned by a previous call to open().
audio_stream_source_t source	Indicates the source that shall be used for the Audio stream.

Description

This ioctl call informs the audio device which source shall be used for the input data. The possible sources are demux or memory. If AUDIO_SOURCE_MEMORY is selected, the data is fed to the Audio Device through the write command.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_MUTE

Name

AUDIO_SET_MUTE

Attention:

This ioctl is deprecated

Synopsis

int **`ioctl`**(int *fd*, *AUDIO_SET_MUTE*, boolean *state*)

Arguments

int fd	File descriptor returned by a previous call to open().
boolean state	Indicates if audio device shall mute or not. TRUE: Audio Mute FALSE: Audio Un-mute

Description

This ioctl is for DVB devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* with the `V4L2_DEC_CMD_START_MUTE_AUDIO` flag instead.

This ioctl call asks the audio device to mute the stream that is currently being played.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_AV_SYNC

Name

AUDIO_SET_AV_SYNC

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_SET_AV_SYNC*, boolean *state*)

Arguments

int fd	File descriptor returned by a previous call to open().
boolean state	Tells the DVB subsystem if A/V synchronization shall be ON or OFF. TRUE: AV-sync ON FALSE: AV-sync OFF

Description

This ioctl call asks the Audio Device to turn ON or OFF A/V synchronization.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_BYPASS_MODE

Name

AUDIO_SET_BYPASS_MODE

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_SET_BYPASS_MODE*, boolean *mode*)

Arguments

int fd	File descriptor returned by a previous call to open().
boolean mode	Enables or disables the decoding of the current Audio stream in the DVB subsystem. TRUE: Bypass is disabled FALSE: Bypass is enabled

Description

This ioctl call asks the Audio Device to bypass the Audio decoder and forward the stream without decoding. This mode shall be used if streams that can't be handled by the DVB system shall be decoded. Dolby DigitalTM streams are automatically forwarded by the DVB subsystem if the hardware can handle it.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_CHANNEL_SELECT

Name

AUDIO_CHANNEL_SELECT

Attention:

This ioctl is deprecated

Synopsis

int **`ioctl`**(int *fd*, *AUDIO_CHANNEL_SELECT*, struct **audio_channel_select*)

Arguments

int fd	File descriptor returned by a previous call to open().
audio_channel_select_t ch	Select the output format of the audio (mono left/right, stereo).

Description

This ioctl is for DVB devices only. To control a V4L2 decoder use the V4L2 `V4L2_CID_MPEG_AUDIO_DEC_PLAYBACK` control instead.

This ioctl call asks the Audio Device to select the requested channel if possible.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_BILINGUAL_CHANNEL_SELECT

Name

AUDIO_BILINGUAL_CHANNEL_SELECT

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_BILINGUAL_CHANNEL_SELECT*, struct *audio_channel_select*)

Arguments

int fd	File descriptor returned by a previous call to open().
audio_channel_select_t ch	Select the output format of the audio (mono left/right, stereo).

Description

This ioctl is obsolete. Do not use in new drivers. It has been replaced by the V4L2 `V4L2_CID_MPEG_AUDIO_DEC_MULTILINGUAL_PLAYBACK` control for MPEG decoders controlled through V4L2.

This ioctl call asks the Audio Device to select the requested channel for bilingual streams if possible.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_GET_PTS

Name

AUDIO_GET_PTS

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_GET_PTS*, __u64 *pts*)

Arguments

int fd	File descriptor returned by a previous call to open().
__u64 *pts	Returns the 33-bit timestamp as defined in ITU T-REC-H.222.0 / ISO/IEC 13818-1. The PTS should belong to the currently played frame if possible, but may also be a value close to it like the PTS of the last decoded frame or the last PTS extracted by the PES parser.

Description

This ioctl is obsolete. Do not use in new drivers. If you need this functionality, then please contact the linux-media mailing list (https://linuxtv.org/lists.php).

This ioctl call asks the Audio Device to return the current PTS timestamp.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_GET_STATUS

Name

AUDIO_GET_STATUS

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_GET_STATUS*, struct *audio_status* **status*)

Arguments

int fd	File descriptor returned by a previous call to open().
struct audio_status *status	Returns the current state of Audio Device.

Description

This ioctl call asks the Audio Device to return the current state of the Audio Device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_GET_CAPABILITIES

Name

AUDIO_GET_CAPABILITIES

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_GET_CAPABILITIES*, unsigned int **cap*)

Arguments

int fd	File descriptor returned by a previous call to open().
unsigned int *cap	Returns a bit array of supported sound formats.

Description

This ioctl call asks the Audio Device to tell us about the decoding capabilities of the audio hardware.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_CLEAR_BUFFER

Name

AUDIO_CLEAR_BUFFER

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_CLEAR_BUFFER*)

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This ioctl call asks the Audio Device to clear all software and hardware buffers of the audio decoder device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_ID

Name

AUDIO_SET_ID

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_SET_ID*, int *id*)

Arguments

int fd	File descriptor returned by a previous call to open().
int id	audio sub-stream id

Description

This ioctl selects which sub-stream is to be decoded if a program or system stream is sent to the video device. If no audio stream type is set the id has to be in [0xC0,0xDF] for MPEG sound, in [0x80,0x87] for AC3 and in [0xA0,0xA7] for LPCM. More specifications may follow for other stream types. If the stream type is set the id just specifies the substream id of the audio stream and only the first 5 bits are recognized.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_MIXER

Name

AUDIO_SET_MIXER

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(int *fd*, *AUDIO_SET_MIXER*, struct *audio_mixer *mix*)

Arguments

int fd	File descriptor returned by a previous call to open().
audio_mixer_t *mix	mixer settings.

Description

This ioctl lets you adjust the mixer settings of the audio decoder.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_STREAMTYPE

Name

AUDIO_SET_STREAMTYPE

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(fd, *AUDIO_SET_STREAMTYPE*, int *type*)

Arguments

int fd	File descriptor returned by a previous call to open().
int type	stream type

Description

This ioctl tells the driver which kind of audio stream to expect. This is useful if the stream offers several audio sub-streams like LPCM and AC3.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	type is not a valid or supported stream type.

AUDIO_SET_EXT_ID

Name

AUDIO_SET_EXT_ID

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(fd, *AUDIO_SET_EXT_ID*, int *id*)

Arguments

int fd	File descriptor returned by a previous call to open().
int id	audio sub_stream_id

Description

This ioctl can be used to set the extension id for MPEG streams in DVD playback. Only the first 3 bits are recognized.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	id is not a valid id.

AUDIO_SET_ATTRIBUTES

Name

AUDIO_SET_ATTRIBUTES

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(fd, *AUDIO_SET_ATTRIBUTES*, struct *audio_attributes* *attr)

Arguments

int fd	File descriptor returned by a previous call to open().
audio_attributes_t attr	audio attributes according to section ??

Description

This ioctl is intended for DVD playback and allows you to set certain information about the audio stream.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	attr is not a valid or supported attribute setting.

AUDIO_SET_KARAOKE

Name

AUDIO_SET_KARAOKE

Attention:

This ioctl is deprecated

Synopsis

int **ioctl**(fd, *AUDIO_SET_KARAOKE*, struct *audio_karaoke* *karaoke)

Arguments

int fd	File descriptor returned by a previous call to open().
audio_karaoke_t *karaoke	karaoke settings according to section ??.

Description

This ioctl allows one to set the mixer settings for a karaoke DVD.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL	karaoke is not a valid or supported karaoke setting.

1.3.8 Examples

In this section we would like to present some examples for using the DVB API.

Note:

This section is out of date, and the code below won't even compile. Please refer to the libdvbv5 for updated/recommended examples.

Example: Tuning

We will start with a generic tuning subroutine that uses the frontend and SEC, as well as the demux devices. The example is given for QPSK tuners, but can easily be adjusted for QAM.

```
#include <sys/ioctl.h>
#include <stdio.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <unistd.h>

#include <linux/dvb/dmx.h>
#include <linux/dvb/frontend.h>
#include <linux/dvb/sec.h>
#include <sys/poll.h>

#define DMX "/dev/dvb/adapter0/demux1"
#define FRONT "/dev/dvb/adapter0/frontend1"
#define SEC "/dev/dvb/adapter0/sec1"

/* routine for checking if we have a signal and other status information*/
int FEReadStatus(int fd, fe_status_t *stat)
{
    int ans;

    if ( (ans = ioctl(fd,FE_READ_STATUS,stat) < 0)){
        perror("FE READ STATUS: ");
        return -1;
    }

    if (*stat & FE_HAS_POWER)
        printf("FE HAS POWER\\n");

    if (*stat & FE_HAS_SIGNAL)
        printf("FE HAS SIGNAL\\n");

    if (*stat & FE_SPECTRUM_INV)
        printf("SPEKTRUM INV\\n");
```

```
    return 0;
}


/* tune qpsk */
/* freq:              frequency of transponder                    */
/* vpid, apid, tpid: PIDs of video, audio and teletext TS packets  */
/* diseqc:            DiSEqC address of the used LNB              */
/* pol:               Polarisation                                */
/* srate:             Symbol Rate                                 */
/* fec.               FEC                                         */
/* lnb_lof1:          local frequency of lower LNB band           */
/* lnb_lof2:          local frequency of upper LNB band           */
/* lnb_slof:          switch frequency of LNB                     */

int set_qpsk_channel(int freq, int vpid, int apid, int tpid,
        int diseqc, int pol, int srate, int fec, int lnb_lof1,
        int lnb_lof2, int lnb_slof)
{
    struct secCommand scmd;
    struct secCmdSequence scmds;
    struct dmx_pes_filter_params pesFilterParams;
    FrontendParameters frp;
    struct pollfd pfd[1];
    FrontendEvent event;
    int demux1, demux2, demux3, front;

    frequency = (uint32_t) freq;
    symbolrate = (uint32_t) srate;

    if((front = open(FRONT,O_RDWR)) < 0){
        perror("FRONTEND DEVICE: ");
        return -1;
    }

    if((sec = open(SEC,O_RDWR)) < 0){
        perror("SEC DEVICE: ");
        return -1;
    }

    if (demux1 < 0){
        if ((demux1=open(DMX, O_RDWR|O_NONBLOCK))
            < 0){
            perror("DEMUX DEVICE: ");
            return -1;
        }
    }

    if (demux2 < 0){
        if ((demux2=open(DMX, O_RDWR|O_NONBLOCK))
            < 0){
            perror("DEMUX DEVICE: ");
            return -1;
        }
    }

    if (demux3 < 0){
        if ((demux3=open(DMX, O_RDWR|O_NONBLOCK))
            < 0){
            perror("DEMUX DEVICE: ");
            return -1;
        }
    }
```

```
    if (freq < lnb_slof) {
        frp.Frequency = (freq - lnb_lof1);
        scmds.continuousTone = SEC_TONE_OFF;
    } else {
        frp.Frequency = (freq - lnb_lof2);
        scmds.continuousTone = SEC_TONE_ON;
    }
    frp.Inversion = INVERSION_AUTO;
    if (pol) scmds.voltage = SEC_VOLTAGE_18;
    else scmds.voltage = SEC_VOLTAGE_13;

    scmd.type=0;
    scmd.u.diseqc.addr=0x10;
    scmd.u.diseqc.cmd=0x38;
    scmd.u.diseqc.numParams=1;
    scmd.u.diseqc.params[0] = 0xF0 | ((diseqc * 4) & 0x0F) |
        (scmds.continuousTone == SEC_TONE_ON ? 1 : 0) |
        (scmds.voltage==SEC_VOLTAGE_18 ? 2 : 0);

    scmds.miniCommand=SEC_MINI_NONE;
    scmds.numCommands=1;
    scmds.commands=&scmd;
    if (ioctl(sec, SEC_SEND_SEQUENCE, &scmds) < 0){
        perror("SEC SEND: ");
        return -1;
    }

    if (ioctl(sec, SEC_SEND_SEQUENCE, &scmds) < 0){
        perror("SEC SEND: ");
        return -1;
    }

    frp.u.qpsk.SymbolRate = srate;
    frp.u.qpsk.FEC_inner = fec;

    if (ioctl(front, FE_SET_FRONTEND, &frp) < 0){
        perror("QPSK TUNE: ");
        return -1;
    }

    pfd[0].fd = front;
    pfd[0].events = POLLIN;

    if (poll(pfd,1,3000)){
        if (pfd[0].revents & POLLIN){
            printf("Getting QPSK event\\n");
            if ( ioctl(front, FE_GET_EVENT, &event)

                 == -EOVERFLOW){
                perror("qpsk get event");
                return -1;
            }
            printf("Received ");
            switch(event.type){
            case FE_UNEXPECTED_EV:
                printf("unexpected event\\n");
                return -1;
            case FE_FAILURE_EV:
                printf("failure event\\n");
                return -1;

            case FE_COMPLETION_EV:
```

```
                printf("completion event\\n");
            }
        }
    }


    pesFilterParams.pid     = vpid;
    pesFilterParams.input   = DMX_IN_FRONTEND;
    pesFilterParams.output  = DMX_OUT_DECODER;
    pesFilterParams.pes_type = DMX_PES_VIDEO;
    pesFilterParams.flags   = DMX_IMMEDIATE_START;
    if (ioctl(demux1, DMX_SET_PES_FILTER, &pesFilterParams) < 0){
        perror("set_vpid");
        return -1;
    }

    pesFilterParams.pid     = apid;
    pesFilterParams.input   = DMX_IN_FRONTEND;
    pesFilterParams.output  = DMX_OUT_DECODER;
    pesFilterParams.pes_type = DMX_PES_AUDIO;
    pesFilterParams.flags   = DMX_IMMEDIATE_START;
    if (ioctl(demux2, DMX_SET_PES_FILTER, &pesFilterParams) < 0){
        perror("set_apid");
        return -1;
    }

    pesFilterParams.pid     = tpid;
    pesFilterParams.input   = DMX_IN_FRONTEND;
    pesFilterParams.output  = DMX_OUT_DECODER;
    pesFilterParams.pes_type = DMX_PES_TELETEXT;
    pesFilterParams.flags   = DMX_IMMEDIATE_START;
    if (ioctl(demux3, DMX_SET_PES_FILTER, &pesFilterParams) < 0){
        perror("set_tpid");
        return -1;
    }

    return has_signal(fds);
}
```

The program assumes that you are using a universal LNB and a standard DiSEqC switch with up to 4 addresses. Of course, you could build in some more checking if tuning was successful and maybe try to repeat the tuning process. Depending on the external hardware, i.e. LNB and DiSEqC switch, and weather conditions this may be necessary.

Example: The DVR device

The following program code shows how to use the DVR device for recording.

```
#include <sys/ioctl.h>
#include <stdio.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <unistd.h>

#include <linux/dvb/dmx.h>
#include <linux/dvb/video.h>
#include <sys/poll.h>
#define DVR "/dev/dvb/adapter0/dvr1"
```

```
#define AUDIO "/dev/dvb/adapter0/audio1"
#define VIDEO "/dev/dvb/adapter0/video1"

#define BUFFY (188*20)
#define MAX_LENGTH (1024*1024*5) /* record 5MB */


/* switch the demuxes to recording, assuming the transponder is tuned */

/* demux1, demux2: file descriptor of video and audio filters */
/* vpid, apid:     PIDs of video and audio channels           */

int switch_to_record(int demux1, int demux2, uint16_t vpid, uint16_t apid)
{
    struct dmx_pes_filter_params pesFilterParams;

    if (demux1 < 0){
        if ((demux1=open(DMX, O_RDWR|O_NONBLOCK))
            < 0){
            perror("DEMUX DEVICE: ");
            return -1;
        }
    }

    if (demux2 < 0){
        if ((demux2=open(DMX, O_RDWR|O_NONBLOCK))
            < 0){
            perror("DEMUX DEVICE: ");
            return -1;
        }
    }

    pesFilterParams.pid = vpid;
    pesFilterParams.input = DMX_IN_FRONTEND;
    pesFilterParams.output = DMX_OUT_TS_TAP;
    pesFilterParams.pes_type = DMX_PES_VIDEO;
    pesFilterParams.flags = DMX_IMMEDIATE_START;
    if (ioctl(demux1, DMX_SET_PES_FILTER, &pesFilterParams) < 0){
        perror("DEMUX DEVICE");
        return -1;
    }
    pesFilterParams.pid = apid;
    pesFilterParams.input = DMX_IN_FRONTEND;
    pesFilterParams.output = DMX_OUT_TS_TAP;
    pesFilterParams.pes_type = DMX_PES_AUDIO;
    pesFilterParams.flags = DMX_IMMEDIATE_START;
    if (ioctl(demux2, DMX_SET_PES_FILTER, &pesFilterParams) < 0){
        perror("DEMUX DEVICE");
        return -1;
    }
    return 0;
}

/* start recording MAX_LENGTH , assuming the transponder is tuned */

/* demux1, demux2: file descriptor of video and audio filters */
/* vpid, apid:     PIDs of video and audio channels           */
int record_dvr(int demux1, int demux2, uint16_t vpid, uint16_t apid)
{
    int i;
    int len;
    int written;
    uint8_t buf[BUFFY];
```

```
    uint64_t length;
    struct pollfd pfd[1];
    int dvr, dvr_out;

    /* open dvr device */
    if ((dvr = open(DVR, O_RDONLY|O_NONBLOCK)) < 0){
            perror("DVR DEVICE");
            return -1;
    }

    /* switch video and audio demuxes to dvr */
    printf ("Switching dvr on\\n");
    i = switch_to_record(demux1, demux2, vpid, apid);
    printf("finished: ");

    printf("Recording %2.0f MB of test file in TS format\\n",
       MAX_LENGTH/(1024.0*1024.0));
    length = 0;

    /* open output file */
    if ((dvr_out = open(DVR_FILE,O_WRONLY|O_CREAT
                 |O_TRUNC, S_IRUSR|S_IWUSR
                 |S_IRGRP|S_IWGRP|S_IROTH|
                 S_IWOTH)) < 0){
        perror("Can't open file for dvr test");
        return -1;
    }

    pfd[0].fd = dvr;
    pfd[0].events = POLLIN;

    /* poll for dvr data and write to file */
    while (length < MAX_LENGTH ) {
        if (poll(pfd,1,1)){
            if (pfd[0].revents & POLLIN){
                len = read(dvr, buf, BUFFY);
                if (len < 0){
                    perror("recording");
                    return -1;
                }
                if (len > 0){
                    written = 0;
                    while (written < len)
                        written +=
                            write (dvr_out,
                               buf, len);
                    length += len;
                    printf("written %2.0f MB\\r",
                       length/1024./1024.);
                }
            }
        }
    }
    return 0;
}
```

1.3.9 DVB Audio Header File

audio.h

```
/*
```

```
 * audio.h
 *
 * Copyright (C) 2000 Ralph  Metzler <ralph@convergence.de>
 *                  & Marcus Metzler <marcus@convergence.de>
 *                    for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Lesser Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 *
 */

#ifndef _DVBAUDIO_H_
#define _DVBAUDIO_H_

#include <linux/types.h>

typedef enum {
        AUDIO_SOURCE_DEMUX, /* Select the demux as the main source */
        AUDIO_SOURCE_MEMORY /* Select internal memory as the main source */
} audio_stream_source_t;

typedef enum {
        AUDIO_STOPPED,      /* Device is stopped */
        AUDIO_PLAYING,      /* Device is currently playing */
        AUDIO_PAUSED        /* Device is paused */
} audio_play_state_t;

typedef enum {
        AUDIO_STEREO,
        AUDIO_MONO_LEFT,
        AUDIO_MONO_RIGHT,
        AUDIO_MONO,
        AUDIO_STEREO_SWAPPED
} audio_channel_select_t;

typedef struct  audio_mixer
 {
        unsigned int volume_left;
        unsigned int volume_right;
  // what else do we need? bass, pass-through, ...
}  audio_mixer_t
;

typedef struct  audio_status
 {
        int                     AV_sync_state;  /* sync audio and video? */
        int                     mute_state;     /* audio is muted */
```

```
        audio_play_state_t     play_state;     /* current playback state */
        audio_stream_source_t  stream_source;  /* current stream source */
        audio_channel_select_t channel_select; /* currently selected channel */
        int                    bypass_mode;    /* pass on audio data to */
         audio_mixer_t
          mixer_state;    /* current mixer state */
}  audio_status_t
;                              /* separate decoder hardware */

typedef
struct  audio_karaoke
 {  /* if Vocal1 or Vocal2 are non-zero, they get mixed  */
        int vocal1;    /* into left and right t at 70% each */
        int vocal2;    /* if both, Vocal1 and Vocal2 are non-zero, Vocal1 gets*/
        int melody;    /* mixed into the left channel and */
                       /* Vocal2 into the right channel at 100% each. */
                       /* if Melody is non-zero, the melody channel gets mixed*/
}  audio_karaoke_t
;     /* into left and right  */

typedef __u16 audio_attributes_t;
/*   bits: descr. */
/*   15-13 audio coding mode (0=ac3, 2=mpeg1, 3=mpeg2ext, 4=LPCM, 6=DTS, */
/*   12    multichannel extension */
/*   11-10 audio type (0=not spec, 1=language included) */
/*    9- 8 audio application mode (0=not spec, 1=karaoke, 2=surround) */
/*    7- 6 Quantization / DRC (mpeg audio: 1=DRC exists)(lpcm: 0=16bit,  */
/*    5- 4 Sample frequency fs (0=48kHz, 1=96kHz) */
/*    2- 0 number of audio channels (n+1 channels) */

/* for GET_CAPABILITIES and SET_FORMAT, the latter should only set one bit */
#define AUDIO_CAP_DTS    1
#define AUDIO_CAP_LPCM   2
#define AUDIO_CAP_MP1    4
#define AUDIO_CAP_MP2    8
#define AUDIO_CAP_MP3   16
#define AUDIO_CAP_AAC   32
#define AUDIO_CAP_OGG   64
#define AUDIO_CAP_SDDS 128
#define AUDIO_CAP_AC3  256

#define  AUDIO_STOP                 _IO(`o', 1)
#define  AUDIO_PLAY                 _IO(`o', 2)
#define  AUDIO_PAUSE                _IO(`o', 3)
#define  AUDIO_CONTINUE             _IO(`o', 4)
#define  AUDIO_SELECT_SOURCE        _IO(`o', 5)
#define  AUDIO_SET_MUTE             _IO(`o', 6)
#define  AUDIO_SET_AV_SYNC          _IO(`o', 7)
#define  AUDIO_SET_BYPASS_MODE      _IO(`o', 8)
#define  AUDIO_CHANNEL_SELECT       _IO(`o', 9)
#define  AUDIO_GET_STATUS           _IOR(`o', 10,  audio_status_t
)

#define  AUDIO_GET_CAPABILITIES     _IOR(`o', 11, unsigned int)
#define  AUDIO_CLEAR_BUFFER         _IO(`o',  12)
#define  AUDIO_SET_ID               _IO(`o', 13)
#define  AUDIO_SET_MIXER            _IOW(`o', 14,  audio_mixer_t
)
```

```
#define  AUDIO_SET_STREAMTYPE        _IO(`o', 15)
#define  AUDIO_SET_EXT_ID            _IO(`o', 16)
#define  AUDIO_SET_ATTRIBUTES        _IOW(`o', 17, audio_attributes_t)
#define  AUDIO_SET_KARAOKE           _IOW(`o', 18,  audio_karaoke_t
)

/**
 *  AUDIO_GET_PTS
 *
 * Read the 33 bit presentation time stamp as defined
 * in ITU T-REC-H.222.0 / ISO/IEC 13818-1.
 *
 * The PTS should belong to the currently played
 * frame if possible, but may also be a value close to it
 * like the PTS of the last decoded frame or the last PTS
 * extracted by the PES parser.
 */
#define  AUDIO_GET_PTS               _IOR(`o', 19, __u64)
#define  AUDIO_BILINGUAL_CHANNEL_SELECT  _IO(`o', 20)

#endif /* _DVBAUDIO_H_ */
```

1.3.10 DVB Conditional Access Header File

ca.h

```
/*
 * ca.h
 *
 * Copyright (C) 2000 Ralph  Metzler <ralph@convergence.de>
 *                  & Marcus Metzler <marcus@convergence.de>
 *                    for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Lesser Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 *
 */

#ifndef _DVBCA_H_
#define _DVBCA_H_

/* slot interface types and info */

typedef struct  ca_slot_info
 {
        int num;               /* slot number */
```

```
        int type;               /* CA interface this slot supports */
#define  CA_CI
            1     /* CI high level interface */
#define  CA_CI_LINK
       2     /* CI link layer level interface */
#define  CA_CI_PHYS
       4     /* CI physical layer level interface */
#define  CA_DESCR
         8     /* built-in descrambler */
#define  CA_SC
          128     /* simple smart card interface */

        unsigned int flags;
#define  CA_CI_MODULE_PRESENT
 1 /* module (or card) inserted */
#define  CA_CI_MODULE_READY
   2
}  ca_slot_info_t
;

/* descrambler types and info */

typedef struct  ca_descr_info
 {
        unsigned int num;          /* number of available descramblers (keys) */
        unsigned int type;         /* type of supported scrambling system */
#define  CA_ECD
           1
#define  CA_NDS
           2
#define  CA_DSS
           4
}  ca_descr_info_t
;

typedef struct  ca_caps
 {
        unsigned int slot_num;     /* total number of CA card and module slots */
        unsigned int slot_type;    /* OR of all supported types */
        unsigned int descr_num;    /* total number of descrambler slots (keys) */
        unsigned int descr_type;   /* OR of all supported types */
}  ca_caps_t
;

/* a message to/from a CI-CAM */
typedef struct  ca_msg
 {
        unsigned int index;
        unsigned int type;
        unsigned int length;
        unsigned char msg[256];
}  ca_msg_t
;

typedef struct  ca_descr
 {
        unsigned int index;
```

```
        unsigned int parity;    /* 0 == even, 1 == odd */
        unsigned char cw[8];
}  ca_descr_t
;

typedef struct  ca_pid
 {
        unsigned int pid;
        int index;              /* -1 == disable*/
}  ca_pid_t
;

#define  CA_RESET           _IO(`o', 128)
#define  CA_GET_CAP         _IOR(`o', 129,  ca_caps_t
)
#define  CA_GET_SLOT_INFO   _IOR(`o', 130,  ca_slot_info_t
)
#define  CA_GET_DESCR_INFO  _IOR(`o', 131,  ca_descr_info_t
)
#define  CA_GET_MSG         _IOR(`o', 132,  ca_msg_t
)
#define  CA_SEND_MSG        _IOW(`o', 133,  ca_msg_t
)
#define  CA_SET_DESCR       _IOW(`o', 134,  ca_descr_t
)
#define  CA_SET_PID         _IOW(`o', 135,  ca_pid_t
)

#endif
```

1.3.11 DVB Demux Header File

dmx.h

```
/*
 * dmx.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *                  & Ralph  Metzler <ralph@convergence.de>
 *                    for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 *
 */
```

```
#ifndef _UAPI_DVBDMX_H_
#define _UAPI_DVBDMX_H_

#include <linux/types.h>
#ifndef __KERNEL__
#include <time.h>
#endif

#define DMX_FILTER_SIZE 16

enum  dmx_output

{
        DMX_OUT_DECODER , /* Streaming directly to decoder. */
        DMX_OUT_TAP ,     /* Output going to a memory buffer */
                          /* (to be retrieved via the read command).*/
        DMX_OUT_TS_TAP ,  /* Output multiplexed into a new TS  */
                          /* (to be retrieved by reading from the */
                          /* logical DVR device).                 */
        DMX_OUT_TSDEMUX_TAP  /* Like TS_TAP but retrieved from the DMX device */
};

typedef enum  dmx_output
  dmx_output_t
;

typedef  dmx_input

{
        DMX_IN_FRONTEND
, /* Input from a front-end device.  */
        DMX_IN_DVR
       /* Input from the logical DVR device.  */
}  dmx_input_t
;

typedef  dmx_ts_pes

{
        DMX_PES_AUDIO0
,
        DMX_PES_VIDEO0
,
        DMX_PES_TELETEXT0
,
        DMX_PES_SUBTITLE0
,
        DMX_PES_PCR0
,

        DMX_PES_AUDIO1
,
        DMX_PES_VIDEO1
,
        DMX_PES_TELETEXT1
,
        DMX_PES_SUBTITLE1
,
```

```
        DMX_PES_PCR1
,
        DMX_PES_AUDIO2
,
        DMX_PES_VIDEO2
,
        DMX_PES_TELETEXT2
,
        DMX_PES_SUBTITLE2
,
        DMX_PES_PCR2
,
        DMX_PES_AUDIO3
,
        DMX_PES_VIDEO3
,
        DMX_PES_TELETEXT3
,
        DMX_PES_SUBTITLE3
,
        DMX_PES_PCR3
,
        DMX_PES_OTHER

}  dmx_pes_type_t
;

#define DMX_PES_AUDIO     DMX_PES_AUDIO0

#define DMX_PES_VIDEO     DMX_PES_VIDEO0

#define DMX_PES_TELETEXT  DMX_PES_TELETEXT0

#define DMX_PES_SUBTITLE  DMX_PES_SUBTITLE0

#define DMX_PES_PCR       DMX_PES_PCR0


typedef struct  dmx_filter

{
        __u8  filter[DMX_FILTER_SIZE];
        __u8  mask[DMX_FILTER_SIZE];
        __u8  mode[DMX_FILTER_SIZE];
}  dmx_filter_t
;

struct  dmx_sct_filter_params

{
        __u16          pid;
         dmx_filter_t
   filter;
        __u32          timeout;
        __u32          flags;
```

```
#define  DMX_CHECK_CRC
       1
#define  DMX_ONESHOT
        2
#define  DMX_IMMEDIATE_START
 4
#define  DMX_KERNEL_CLIENT
   0x8000
};

struct  dmx_pes_filter_params

{
        __u16          pid;
         dmx_input_t
    input;
         dmx_output_t
   output;
         dmx_pes_type_t
 pes_type;
        __u32          flags;
};

typedef struct  dmx_caps
 {
        __u32 caps;
        int num_decoders;
}  dmx_caps_t
;

typedef  dmx_source
 {
         DMX_SOURCE_FRONT0
 = 0,
         DMX_SOURCE_FRONT1
,
         DMX_SOURCE_FRONT2
,
         DMX_SOURCE_FRONT3
,
         DMX_SOURCE_DVR0
   = 16,
         DMX_SOURCE_DVR1
,
         DMX_SOURCE_DVR2
,
         DMX_SOURCE_DVR3

}  dmx_source_t
;

struct  dmx_stc
 {
        unsigned int num;       /* input : which STC? 0..N */
        unsigned int base;      /* output: divisor for stc to get 90 kHz clock */
        __u64 stc;              /* output: stc in `base'*90 kHz units */
};
```

```
#define  DMX_START                _IO(`o', 41)
#define  DMX_STOP                 _IO(`o', 42)
#define  DMX_SET_FILTER           _IOW(`o', 43, struct  dmx_sct_filter_params
)
#define  DMX_SET_PES_FILTER       _IOW(`o', 44, struct  dmx_pes_filter_params
)
#define  DMX_SET_BUFFER_SIZE      _IO(`o', 45)
#define  DMX_GET_PES_PIDS         _IOR(`o', 47, __u16[5])
#define  DMX_GET_CAPS             _IOR(`o', 48,  dmx_caps_t
)
#define  DMX_SET_SOURCE           _IOW(`o', 49,  dmx_source_t
)
#define  DMX_GET_STC              _IOWR(`o', 50, struct  dmx_stc
)
#define  DMX_ADD_PID              _IOW(`o', 51, __u16)
#define  DMX_REMOVE_PID           _IOW(`o', 52, __u16)

#endif /* _UAPI_DVBDMX_H_ */
```

1.3.12 DVB Frontend Header File

frontend.h

```
/*
 * frontend.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *                    Ralph  Metzler <ralph@convergence.de>
 *                    Holger Waechtler <holger@convergence.de>
 *                    Andre Draszik <ad@convergence.de>
 *                    for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 *
 */

#ifndef _DVBFRONTEND_H_
#define _DVBFRONTEND_H_

#include <linux/types.h>

enum  fe_type
 {
        FE_QPSK ,
        FE_QAM ,
```

```
        FE_OFDM ,
        FE_ATSC
};

enum  fe_caps
 {
        FE_IS_STUPID                    = 0,
        FE_CAN_INVERSION_AUTO           = 0x1,
        FE_CAN_FEC_1_2                  = 0x2,
        FE_CAN_FEC_2_3                  = 0x4,
        FE_CAN_FEC_3_4                  = 0x8,
        FE_CAN_FEC_4_5                  = 0x10,
        FE_CAN_FEC_5_6                  = 0x20,
        FE_CAN_FEC_6_7                  = 0x40,
        FE_CAN_FEC_7_8                  = 0x80,
        FE_CAN_FEC_8_9                  = 0x100,
        FE_CAN_FEC_AUTO                 = 0x200,
        FE_CAN_QPSK                     = 0x400,
        FE_CAN_QAM_16                   = 0x800,
        FE_CAN_QAM_32                   = 0x1000,
        FE_CAN_QAM_64                   = 0x2000,
        FE_CAN_QAM_128                  = 0x4000,
        FE_CAN_QAM_256                  = 0x8000,
        FE_CAN_QAM_AUTO                 = 0x10000,
        FE_CAN_TRANSMISSION_MODE_AUTO   = 0x20000,
        FE_CAN_BANDWIDTH_AUTO           = 0x40000,
        FE_CAN_GUARD_INTERVAL_AUTO      = 0x80000,
        FE_CAN_HIERARCHY_AUTO           = 0x100000,
        FE_CAN_8VSB                     = 0x200000,
        FE_CAN_16VSB                    = 0x400000,
        FE_HAS_EXTENDED_CAPS            = 0x800000,   /* We need more bitspace for newer API
dicate this. */
        FE_CAN_MULTISTREAM              = 0x4000000,  /* frontend supports multi-
stream filtering */
        FE_CAN_TURBO_FEC                = 0x8000000,  /* frontend sup-
ports ``turbo fec modulation'' */
        FE_CAN_2G_MODULATION            = 0x10000000, /* frontend supports ``2nd gen-
eration modulation'' (DVB-S2) */
        FE_NEEDS_BENDING                = 0x20000000, /* not supported any-
more, don't use (frontend requires frequency bending) */
        FE_CAN_RECOVER                  = 0x40000000, /* frontend can re-
cover from a cable unplug automatically */
        FE_CAN_MUTE_TS                  = 0x80000000  /* frontend can stop spuri-
ous TS data output */
};

struct  dvb_frontend_info
 {
        char       name[128];
        enum  fe_type
 type;      /* DEPRECATED. Use  DTV_ENUM_DELSYS  instead */
        __u32      frequency_min;
        __u32      frequency_max;
        __u32      frequency_stepsize;
        __u32      frequency_tolerance;
        __u32      symbol_rate_min;
        __u32      symbol_rate_max;
        __u32      symbol_rate_tolerance;       /* ppm */
```

```
        __u32      notifier_delay;             /* DEPRECATED */
        enum  fe_caps
 caps;
};

/**
 *  Check out the DiSEqC bus spec available on http://www.eutelsat.org/ for
 *  the meaning of this struct...
 */
struct  dvb_diseqc_master_cmd
 {
        __u8 msg [6];   /*  { framing, address, command, data [3] } */
        __u8 msg_len;   /*  valid values are 3...6  */
};

struct  dvb_diseqc_slave_reply
 {
        __u8 msg [4];   /*  { framing, data [3] } */
        __u8 msg_len;   /*  valid values are 0...4, 0 means no msg  */
        int  timeout;   /*  return from ioctl after timeout ms with */
};                      /*  errorcode when no message was received  */

enum  fe_sec_voltage
 {
         SEC_VOLTAGE_13 ,
         SEC_VOLTAGE_18 ,
         SEC_VOLTAGE_OFF
};

enum  fe_sec_tone_mode
 {
         SEC_TONE_ON ,
         SEC_TONE_OFF
};

enum  fe_sec_mini_cmd
 {
         SEC_MINI_A ,
         SEC_MINI_B
};

/**
 * enum  fe_status
 - enumerates the possible frontend status
 * @ FE_HAS_SIGNAL :      found something above the noise level
 * @ FE_HAS_CARRIER :     found a DVB signal
 * @ FE_HAS_VITERBI :     FEC is stable
 * @ FE_HAS_SYNC :        found sync bytes
 * @ FE_HAS_LOCK :        everything's working
 * @ FE_TIMEDOUT :        no lock within the last ~2 seconds
 * @ FE_REINIT :          frontend was reinitialized, application is recommended
 *                      to reset DiSEqC, tone and parameters
 */
enum  fe_status
 {
         FE_HAS_SIGNAL           = 0x01,
         FE_HAS_CARRIER          = 0x02,
         FE_HAS_VITERBI          = 0x04,
```

```
        FE_HAS_SYNC             = 0x08,
        FE_HAS_LOCK             = 0x10,
        FE_TIMEDOUT             = 0x20,
        FE_REINIT               = 0x40,
};

enum  fe_spectral_inversion
 {
        INVERSION_OFF ,
        INVERSION_ON ,
        INVERSION_AUTO
};

enum  fe_code_rate
 {
        FEC_NONE  = 0,
        FEC_1_2 ,
        FEC_2_3 ,
        FEC_3_4 ,
        FEC_4_5 ,
        FEC_5_6 ,
        FEC_6_7 ,
        FEC_7_8 ,
        FEC_8_9 ,
        FEC_AUTO ,
        FEC_3_5 ,
        FEC_9_10 ,
        FEC_2_5 ,
};

enum  fe_modulation
 {
        QPSK ,
        QAM_16 ,
        QAM_32 ,
        QAM_64 ,
        QAM_128 ,
        QAM_256 ,
        QAM_AUTO ,
        VSB_8 ,
        VSB_16 ,
        PSK_8 ,
        APSK_16 ,
        APSK_32 ,
        DQPSK ,
        QAM_4_NR ,
};

enum  fe_transmit_mode
 {
        TRANSMISSION_MODE_2K ,
        TRANSMISSION_MODE_8K ,
        TRANSMISSION_MODE_AUTO ,
        TRANSMISSION_MODE_4K ,
        TRANSMISSION_MODE_1K ,
        TRANSMISSION_MODE_16K ,
        TRANSMISSION_MODE_32K ,
        TRANSMISSION_MODE_C1 ,
```

```
        TRANSMISSION_MODE_C3780 ,
};

enum  fe_guard_interval
 {
        GUARD_INTERVAL_1_32 ,
        GUARD_INTERVAL_1_16 ,
        GUARD_INTERVAL_1_8 ,
        GUARD_INTERVAL_1_4 ,
        GUARD_INTERVAL_AUTO ,
        GUARD_INTERVAL_1_128 ,
        GUARD_INTERVAL_19_128 ,
        GUARD_INTERVAL_19_256 ,
        GUARD_INTERVAL_PN420 ,
        GUARD_INTERVAL_PN595 ,
        GUARD_INTERVAL_PN945 ,
};

enum  fe_hierarchy
 {
        HIERARCHY_NONE ,
        HIERARCHY_1 ,
        HIERARCHY_2 ,
        HIERARCHY_4 ,
        HIERARCHY_AUTO
};

enum  fe_interleaving
 {
        INTERLEAVING_NONE ,
        INTERLEAVING_AUTO ,
        INTERLEAVING_240 ,
        INTERLEAVING_720 ,
};

/* S2API Commands */
#define  DTV_UNDEFINED           0
#define  DTV_TUNE                1
#define  DTV_CLEAR               2
#define  DTV_FREQUENCY           3
#define  DTV_MODULATION          4
#define  DTV_BANDWIDTH_HZ        5
#define  DTV_INVERSION           6
#define  DTV_DISEQC_MASTER       7
#define  DTV_SYMBOL_RATE         8
#define  DTV_INNER_FEC           9
#define  DTV_VOLTAGE             10
#define  DTV_TONE                11
#define  DTV_PILOT               12
#define  DTV_ROLLOFF             13
#define  DTV_DISEQC_SLAVE_REPLY  14

/* Basic enumeration set for querying unlimited capabilities */
#define  DTV_FE_CAPABILITY_COUNT  15
#define  DTV_FE_CAPABILITY        16
#define  DTV_DELIVERY_SYSTEM      17

/* ISDB-T and ISDB-Tsb */
```

```
#define  DTV_ISDBT_PARTIAL_RECEPTION      18
#define  DTV_ISDBT_SOUND_BROADCASTING     19

#define  DTV_ISDBT_SB_SUBCHANNEL_ID       20
#define  DTV_ISDBT_SB_SEGMENT_IDX         21
#define  DTV_ISDBT_SB_SEGMENT_COUNT       22

#define  DTV_ISDBT_LAYERA_FEC                    23
#define  DTV_ISDBT_LAYERA_MODULATION             24
#define  DTV_ISDBT_LAYERA_SEGMENT_COUNT          25
#define  DTV_ISDBT_LAYERA_TIME_INTERLEAVING      26

#define  DTV_ISDBT_LAYERB_FEC                    27
#define  DTV_ISDBT_LAYERB_MODULATION             28
#define  DTV_ISDBT_LAYERB_SEGMENT_COUNT          29
#define  DTV_ISDBT_LAYERB_TIME_INTERLEAVING      30

#define  DTV_ISDBT_LAYERC_FEC                    31
#define  DTV_ISDBT_LAYERC_MODULATION             32
#define  DTV_ISDBT_LAYERC_SEGMENT_COUNT          33
#define  DTV_ISDBT_LAYERC_TIME_INTERLEAVING      34

#define  DTV_API_VERSION          35

#define  DTV_CODE_RATE_HP         36
#define  DTV_CODE_RATE_LP         37
#define  DTV_GUARD_INTERVAL       38
#define  DTV_TRANSMISSION_MODE    39
#define  DTV_HIERARCHY            40

#define  DTV_ISDBT_LAYER_ENABLED  41

#define  DTV_STREAM_ID            42
#define DTV_ISDBS_TS_ID_LEGACY   DTV_STREAM_ID
#define  DTV_DVBT2_PLP_ID_LEGACY  43

#define  DTV_ENUM_DELSYS          44

/* ATSC-MH */
#define  DTV_ATSCMH_FIC_VER              45
#define  DTV_ATSCMH_PARADE_ID            46
#define  DTV_ATSCMH_NOG                  47
#define  DTV_ATSCMH_TNOG                 48
#define  DTV_ATSCMH_SGN                  49
#define  DTV_ATSCMH_PRC                  50
#define  DTV_ATSCMH_RS_FRAME_MODE        51
#define  DTV_ATSCMH_RS_FRAME_ENSEMBLE    52
#define  DTV_ATSCMH_RS_CODE_MODE_PRI     53
#define  DTV_ATSCMH_RS_CODE_MODE_SEC     54
#define  DTV_ATSCMH_SCCC_BLOCK_MODE      55
#define  DTV_ATSCMH_SCCC_CODE_MODE_A     56
#define  DTV_ATSCMH_SCCC_CODE_MODE_B     57
#define  DTV_ATSCMH_SCCC_CODE_MODE_C     58
#define  DTV_ATSCMH_SCCC_CODE_MODE_D     59

#define  DTV_INTERLEAVING                        60
#define  DTV_LNA                                 61
```

```
/* Quality parameters */
#define  DTV_STAT_SIGNAL_STRENGTH         62
#define  DTV_STAT_CNR                     63
#define  DTV_STAT_PRE_ERROR_BIT_COUNT     64
#define  DTV_STAT_PRE_TOTAL_BIT_COUNT     65
#define  DTV_STAT_POST_ERROR_BIT_COUNT    66
#define  DTV_STAT_POST_TOTAL_BIT_COUNT    67
#define  DTV_STAT_ERROR_BLOCK_COUNT       68
#define  DTV_STAT_TOTAL_BLOCK_COUNT       69

#define DTV_MAX_COMMAND          DTV_STAT_TOTAL_BLOCK_COUNT

enum  fe_pilot
 {
        PILOT_ON ,
        PILOT_OFF ,
        PILOT_AUTO ,
};

enum  fe_rolloff
 {
        ROLLOFF_35 , /* Implied value in DVB-S, default for DVB-S2 */
        ROLLOFF_20 ,
        ROLLOFF_25 ,
        ROLLOFF_AUTO ,
};

enum  fe_delivery_system
 {
        SYS_UNDEFINED ,
        SYS_DVBC_ANNEX_A ,
        SYS_DVBC_ANNEX_B ,
        SYS_DVBT ,
        SYS_DSS ,
        SYS_DVBS ,
        SYS_DVBS2 ,
        SYS_DVBH ,
        SYS_ISDBT ,
        SYS_ISDBS ,
        SYS_ISDBC ,
        SYS_ATSC ,
        SYS_ATSCMH ,
        SYS_DTMB ,
        SYS_CMMB ,
        SYS_DAB ,
        SYS_DVBT2 ,
        SYS_TURBO ,
        SYS_DVBC_ANNEX_C ,
};

/* backward compatibility */
#define SYS_DVBC_ANNEX_AC        SYS_DVBC_ANNEX_A
#define SYS_DMBTH  SYS_DTMB  /* DMB-TH is legacy name, use DTMB instead */

/* ATSC-MH */

enum  atscmh_sccc_block_mode
 {
```

```
        ATSCMH_SCCC_BLK_SEP       = 0,
        ATSCMH_SCCC_BLK_COMB      = 1,
        ATSCMH_SCCC_BLK_RES       = 2,
};

enum  atscmh_sccc_code_mode
 {
        ATSCMH_SCCC_CODE_HLF      = 0,
        ATSCMH_SCCC_CODE_QTR      = 1,
        ATSCMH_SCCC_CODE_RES      = 2,
};

enum  atscmh_rs_frame_ensemble
 {
        ATSCMH_RSFRAME_ENS_PRI    = 0,
        ATSCMH_RSFRAME_ENS_SEC    = 1,
};

enum  atscmh_rs_frame_mode
 {
        ATSCMH_RSFRAME_PRI_ONLY   = 0,
        ATSCMH_RSFRAME_PRI_SEC    = 1,
        ATSCMH_RSFRAME_RES        = 2,
};

enum  atscmh_rs_code_mode
 {
        ATSCMH_RSCODE_211_187     = 0,
        ATSCMH_RSCODE_223_187     = 1,
        ATSCMH_RSCODE_235_187     = 2,
        ATSCMH_RSCODE_RES         = 3,
};

#define  NO_STREAM_ID_FILTER      (~0U)
#define  LNA_AUTO                 (~0U)

struct dtv_cmds_h {
        char    *name;          /* A display name for debugging purposes */

        __u32   cmd;            /* A unique ID */

        /* Flags */
        __u32   set:1;          /* Either a set or get property */
        __u32   buffer:1;       /* Does this property use the buffer? */
        __u32   reserved:30;    /* Align */
};

/**
 * Scale types for the quality parameters.
 * @:ref:FE_SCALE_NOT_AVAILABLE <frontend-stat-properties>: That QoS mea-
sure is not available. That
 *                           could indicate a temporary or a permanent
 *                           condition.
 * @:ref:FE_SCALE_DECIBEL <frontend-stat-properties>: The scale is mea-
sured in 0.001 dB steps, typically
 *                used on signal measures.
 * @:ref:FE_SCALE_RELATIVE <frontend-stat-properties>: The scale is a relative per-
centual measure,
```

```
 *                       ranging from 0 (0%) to 0xffff (100%).
 * @:ref:FE_SCALE_COUNTER <frontend-stat-properties>: The scale counts the occur-
rence of an event, like
 *                       bit error, block error, lapsed time.
 */
 fecap_scale_params  {
         FE_SCALE_NOT_AVAILABLE  = 0,
         FE_SCALE_DECIBEL ,
         FE_SCALE_RELATIVE ,
         FE_SCALE_COUNTER
};

/**
 * struct  dtv_stats
 - Used for reading a DTV status property
 *
 * @value:      value of the measure. Should range from 0 to 0xffff;
 * @scale:      Filled with  fecap_scale_params  - the scale
 *              in usage for that parameter
 *
 * For most delivery systems, this will return a single value for each
 * parameter.
 * It should be noticed, however, that new OFDM delivery systems like
 * ISDB can use different modulation types for each group of carriers.
 * On such standards, up to 8 groups of statistics can be provided, one
 * for each carrier group (called ``layer'' on ISDB).
 * In order to be consistent with other delivery systems, the first
 * value refers to the entire set of carriers (``global'').
 * dtv_status:scale should use the value  FE_SCALE_NOT_AVAILABLE  when
 * the value for the entire group of carriers or from one specific layer
 * is not provided by the hardware.
 * st.len should be filled with the latest filled status + 1.
 *
 * In other words, for ISDB, those values should be filled like:
 *      u.st.stat.svalue[0] = global statistics;
 *      u.st.stat.scale[0] =  FE_SCALE_DECIBEL ;
 *      u.st.stat.value[1] = layer A statistics;
 *      u.st.stat.scale[1] =  FE_SCALE_NOT_AVAILABLE  (if not available);
 *      u.st.stat.svalue[2] = layer B statistics;
 *      u.st.stat.scale[2] =  FE_SCALE_DECIBEL ;
 *      u.st.stat.svalue[3] = layer C statistics;
 *      u.st.stat.scale[3] =  FE_SCALE_DECIBEL ;
 *      u.st.len = 4;
 */
struct  dtv_stats
 {
        __u8 scale;     /*  fecap_scale_params  type */
        union {
                __u64 uvalue;   /* for counters and relative scales */
                __s64 svalue;   /* for 0.001 dB measures */
        };
} __attribute__ ((packed));

#define MAX_DTV_STATS   4

struct  dtv_fe_stats
 {
        __u8 len;
```

```
        struct  dtv_stats
 stat[MAX_DTV_STATS];
} __attribute__ ((packed));

struct  dtv_property
 {
        __u32 cmd;
        __u32 reserved[3];
        union {
                __u32 data;
                struct  dtv_fe_stats
 st;
                struct {
                        __u8 data[32];
                        __u32 len;
                        __u32 reserved1[3];
                        void *reserved2;
                } buffer;
        } u;
        int result;
} __attribute__ ((packed));

/* num of properties cannot exceed DTV_IOCTL_MAX_MSGS per ioctl */
#define DTV_IOCTL_MAX_MSGS 64

struct  dtv_properties
 {
        __u32 num;
        struct  dtv_property
 *props;
};

#if defined(__DVB_CORE__) || !defined (__KERNEL__)

/*
 * DEPRECATED: The DVBv3 ioctls, structs and enums should not be used on
 * newer programs, as it doesn't support the second generation of digital
 * TV standards, nor supports newer delivery systems.
 */

enum  fe_bandwidth
 {
         BANDWIDTH_8_MHZ ,
         BANDWIDTH_7_MHZ ,
         BANDWIDTH_6_MHZ ,
         BANDWIDTH_AUTO ,
         BANDWIDTH_5_MHZ ,
         BANDWIDTH_10_MHZ ,
         BANDWIDTH_1_712_MHZ ,
};

/* This is needed for legacy userspace support */
typedef enum  fe_sec_voltage
  fe_sec_voltage_t
;
typedef enum  fe_caps
  fe_caps_t
;
```

```
typedef enum  fe_type
  fe_type_t
;
typedef enum  fe_sec_tone_mode
  fe_sec_tone_mode_t
;
typedef enum  fe_sec_mini_cmd
  fe_sec_mini_cmd_t
;
typedef enum  fe_status
  fe_status_t
;
typedef enum  fe_spectral_inversion
  fe_spectral_inversion_t
;
typedef enum  fe_code_rate
  fe_code_rate_t
;
typedef enum  fe_modulation
  fe_modulation_t
;
typedef enum  fe_transmit_mode
  fe_transmit_mode_t
;
typedef enum  fe_bandwidth
  fe_bandwidth_t
;
typedef enum  fe_guard_interval
  fe_guard_interval_t
;
typedef enum  fe_hierarchy
  fe_hierarchy_t
;
typedef enum  fe_pilot
  fe_pilot_t
;
typedef enum  fe_rolloff
  fe_rolloff_t
;
typedef enum  fe_delivery_system
  fe_delivery_system_t
;

struct  dvb_qpsk_parameters
 {
        __u32           symbol_rate;  /* symbol rate in Symbols per second */
         fe_code_rate_t
  fec_inner;    /* forward error correction (see above) */
};

struct  dvb_qam_parameters
 {
        __u32           symbol_rate; /* symbol rate in Symbols per second */
         fe_code_rate_t
  fec_inner;   /* forward error correction (see above) */
         fe_modulation_t
 modulation;  /* modulation type (see above) */
};
```

```
struct  dvb_vsb_parameters
 {
         fe_modulation_t
 modulation;  /* modulation type (see above) */
};

struct  dvb_ofdm_parameters
 {
         fe_bandwidth_t
      bandwidth;
         fe_code_rate_t
      code_rate_HP;  /* high priority stream code rate */
         fe_code_rate_t
      code_rate_LP;  /* low priority stream code rate */
         fe_modulation_t
     constellation; /* modulation type (see above) */
         fe_transmit_mode_t
  transmission_mode;
         fe_guard_interval_t
 guard_interval;
         fe_hierarchy_t
      hierarchy_information;
};

struct  dvb_frontend_parameters
 {
        __u32 frequency;     /* (absolute) frequency in Hz for DVB-C/DVB-T/ATSC */
                             /* intermediate frequency in kHz for DVB-S */
         fe_spectral_inversion_t
 inversion;
        union {
                struct  dvb_qpsk_parameters
 qpsk;         /* DVB-S */
                struct  dvb_qam_parameters
  qam;          /* DVB-C */
                struct  dvb_ofdm_parameters
 ofdm;         /* DVB-T */
                struct  dvb_vsb_parameters
 vsb;           /* ATSC */
        } u;
};

struct  dvb_frontend_event
 {
         fe_status_t
 status;
        struct  dvb_frontend_parameters
 parameters;
};
#endif

#define  FE_SET_PROPERTY
           _IOW(`o', 82, struct  dtv_properties
)
#define  FE_GET_PROPERTY             _IOR(`o', 83, struct  dtv_properties
)
```

```
/**
 * When set, this flag will disable any zigzagging or other ``normal'' tuning
 * behaviour. Additionally, there will be no automatic monitoring of the lock
 * status, and hence no frontend events will be generated. If a frontend device
 * is closed, this flag will be automatically turned off when the device is
 * reopened read-write.
 */
#define  FE_TUNE_MODE_ONESHOT
 0x01

#define  FE_GET_INFO                _IOR(`o', 61, struct  dvb_frontend_info
)

#define  FE_DISEQC_RESET_OVERLOAD    _IO(`o', 62)
#define  FE_DISEQC_SEND_MASTER_CMD   _IOW(`o', 63, struct  dvb_diseqc_master_cmd
)
#define  FE_DISEQC_RECV_SLAVE_REPLY  _IOR(`o', 64, struct  dvb_diseqc_slave_reply
)
#define  FE_DISEQC_SEND_BURST        _IO(`o', 65)  /*  fe_sec_mini_cmd_t
 */

#define  FE_SET_TONE                 _IO(`o', 66)  /*  fe_sec_tone_mode_t
 */
#define  FE_SET_VOLTAGE              _IO(`o', 67)  /*  fe_sec_voltage_t
 */
#define  FE_ENABLE_HIGH_LNB_VOLTAGE  _IO(`o', 68)  /* int */

#define  FE_READ_STATUS              _IOR(`o', 69,  fe_status_t
)
#define  FE_READ_BER                 _IOR(`o', 70, __u32)
#define  FE_READ_SIGNAL_STRENGTH     _IOR(`o', 71, __u16)
#define  FE_READ_SNR                 _IOR(`o', 72, __u16)
#define  FE_READ_UNCORRECTED_BLOCKS  _IOR(`o', 73, __u32)

#define  FE_SET_FRONTEND             _IOW(`o', 76, struct  dvb_frontend_parameters
)
#define  FE_GET_FRONTEND             _IOR(`o', 77, struct  dvb_frontend_parameters
)
#define  FE_SET_FRONTEND_TUNE_MODE   _IO(`o', 81) /* unsigned int */
#define  FE_GET_EVENT                _IOR(`o', 78, struct  dvb_frontend_event
)

#define  FE_DISHNETWORK_SEND_LEGACY_CMD  _IO(`o', 80) /* unsigned int */

#endif /*_DVBFRONTEND_H_*/
```

1.3.13 DVB Network Header File

net.h

```
/*
 * net.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *                  & Ralph  Metzler <ralph@convergence.de>
 *                    for convergence integrated media GmbH
 *
```

```
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 *
 */

#ifndef _DVBNET_H_
#define _DVBNET_H_

#include <linux/types.h>

struct  dvb_net_if
 {
        __u16 pid;
        __u16 if_num;
        __u8  feedtype;
#define  DVB_NET_FEEDTYPE_MPE
 0  /* multi protocol encapsulation */
#define  DVB_NET_FEEDTYPE_ULE
 1  /* ultra lightweight encapsulation */
};

#define  NET_ADD_IF     _IOWR(`o', 52, struct  dvb_net_if
)
#define  NET_REMOVE_IF  _IO(`o', 53)
#define  NET_GET_IF     _IOWR(`o', 54, struct  dvb_net_if
)

/* binary compatibility cruft: */
struct __dvb_net_if_old {
        __u16 pid;
        __u16 if_num;
};
#define __NET_ADD_IF_OLD _IOWR(`o', 52, struct __dvb_net_if_old)
#define __NET_GET_IF_OLD _IOWR(`o', 54, struct __dvb_net_if_old)

#endif /*_DVBNET_H_*/
```

1.3.14 DVB Video Header File

video.h

```
/*
 * video.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *                  & Ralph  Metzler <ralph@convergence.de>
```

```
 *                    for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
 *
 */

#ifndef _UAPI_DVBVIDEO_H_
#define _UAPI_DVBVIDEO_H_

#include <linux/types.h>
#ifndef __KERNEL__
#include <time.h>
#endif

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;

typedef enum {
         VIDEO_SYSTEM_PAL,
         VIDEO_SYSTEM_NTSC,
         VIDEO_SYSTEM_PALN,
         VIDEO_SYSTEM_PALNc,
         VIDEO_SYSTEM_PALM,
         VIDEO_SYSTEM_NTSC60,
         VIDEO_SYSTEM_PAL60,
         VIDEO_SYSTEM_PALM60
} video_system_t;

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;

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

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;

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

/* Decoder commands */
#define VIDEO_CMD_PLAY        (0)
#define VIDEO_CMD_STOP        (1)
#define VIDEO_CMD_FREEZE      (2)
#define VIDEO_CMD_CONTINUE    (3)

/* Flags for VIDEO_CMD_FREEZE */
#define VIDEO_CMD_FREEZE_TO_BLACK       (1 << 0)

/* Flags for VIDEO_CMD_STOP */
#define VIDEO_CMD_STOP_TO_BLACK         (1 << 0)
#define VIDEO_CMD_STOP_IMMEDIATELY      (1 << 1)

/* Play input formats: */
/* The decoder has no special format requirements */
#define VIDEO_PLAY_FMT_NONE         (0)
/* The decoder requires full GOPs */
#define VIDEO_PLAY_FMT_GOP          (1)

/* 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 nor-
mal speed. */
                        __s32 speed;
                        __u32 format;
                } play;

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

/* FIELD_UNKNOWN can be used if the hardware does not know whether
```

```
   the Vsync is for an odd, even or progressive (i.e. non-interlaced)
   field. */
#define VIDEO_VSYNC_FIELD_UNKNOWN       (0)
#define VIDEO_VSYNC_FIELD_ODD           (1)
#define VIDEO_VSYNC_FIELD_EVEN          (2)
#define VIDEO_VSYNC_FIELD_PROGRESSIVE   (3)

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
        __kernel_time_t timestamp;
        union {
                video_size_t size;
                unsigned int frame_rate;        /* in frames per 1000sec */
                unsigned char vsync_field;      /* unknown/odd/even/progressive */
        } u;
};

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 */
};

struct  video_still_picture
 {
        char __user *iFrame;        /* pointer to a single iframe in memory */
        __s32 size;
};

typedef
struct  video_highlight
 {
        int     active;      /*    1=show highlight, 0=hide highlight */
        __u8    contrast1;   /*    7- 4  Pattern pixel contrast */
                             /*    3- 0  Background pixel contrast */
        __u8    contrast2;   /*    7- 4  Emphasis pixel-2 contrast */
                             /*    3- 0  Emphasis pixel-1 contrast */
        __u8    color1;      /*    7- 4  Pattern pixel color */
                             /*    3- 0  Background pixel color */
        __u8    color2;      /*    7- 4  Emphasis pixel-2 color */
                             /*    3- 0  Emphasis pixel-1 color */
        __u32    ypos;       /*   23-22  auto action mode */
                             /*   21-12  start y */
                             /*    9- 0  end y */
        __u32    xpos;       /*   23-22  button color number */
                             /*   21-12  start x */
                             /*    9- 0  end x */
}  video_highlight_t
;
```

```
typedef struct  video_spu
 {
        int active;
        int stream_id;
}  video_spu_t
;

typedef struct  video_spu_palette
 {      /* SPU Palette information */
        int length;
        __u8 __user *palette;
}  video_spu_palette_t
;

typedef struct  video_navi_pack
 {
        int length;          /* 0 ... 1024 */
        __u8 data[1024];
}  video_navi_pack_t
;

typedef __u16 video_attributes_t;
/*   bits: descr. */
/*   15-14 Video compression mode (0=MPEG-1, 1=MPEG-2) */
/*   13-12 TV system (0=525/60, 1=625/50) */
/*   11-10 Aspect ratio (0=4:3, 3=16:9) */
/*    9- 8 permitted display mode on 4:3 monitor (0=both, 1=only pan-sca */
/*    7    line 21-1 data present in GOP (1=yes, 0=no) */
/*    6    line 21-2 data present in GOP (1=yes, 0=no) */
/*    5- 3 source resolution (0=720x480/576, 1=704x480/576, 2=352x480/57 */
/*    2    source letterboxed (1=yes, 0=no) */
/*    0    film/camera mode (0=
 *camera, 1=film (625/50 only)) */

/* 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

#define  VIDEO_STOP                 _IO(`o', 21)
#define  VIDEO_PLAY                 _IO(`o', 22)
#define  VIDEO_FREEZE               _IO(`o', 23)
#define  VIDEO_CONTINUE             _IO(`o', 24)
#define  VIDEO_SELECT_SOURCE        _IO(`o', 25)
#define  VIDEO_SET_BLANK            _IO(`o', 26)
#define  VIDEO_GET_STATUS           _IOR(`o', 27, struct  video_status
)
#define  VIDEO_GET_EVENT            _IOR(`o', 28, struct  video_event
```

```
)
#define  VIDEO_SET_DISPLAY_FORMAT    _IO(`o', 29)
#define  VIDEO_STILLPICTURE          _IOW(`o', 30, struct  video_still_picture
)
#define  VIDEO_FAST_FORWARD          _IO(`o', 31)
#define  VIDEO_SLOWMOTION            _IO(`o', 32)
#define  VIDEO_GET_CAPABILITIES      _IOR(`o', 33, unsigned int)
#define  VIDEO_CLEAR_BUFFER          _IO(`o',  34)
#define  VIDEO_SET_ID                _IO(`o', 35)
#define  VIDEO_SET_STREAMTYPE        _IO(`o', 36)
#define  VIDEO_SET_FORMAT            _IO(`o', 37)
#define  VIDEO_SET_SYSTEM            _IO(`o', 38)
#define  VIDEO_SET_HIGHLIGHT         _IOW(`o', 39,  video_highlight_t
)
#define  VIDEO_SET_SPU               _IOW(`o', 50,  video_spu_t
)
#define  VIDEO_SET_SPU_PALETTE       _IOW(`o', 51,  video_spu_palette_t
)
#define  VIDEO_GET_NAVI              _IOR(`o', 52,  video_navi_pack_t
)
#define  VIDEO_SET_ATTRIBUTES        _IO(`o', 53)
#define  VIDEO_GET_SIZE              _IOR(`o', 55, video_size_t)
#define  VIDEO_GET_FRAME_RATE        _IOR(`o', 56, unsigned int)

/**
 *  VIDEO_GET_PTS
 *
 * Read the 33 bit presentation time stamp as defined
 * in ITU T-REC-H.222.0 / ISO/IEC 13818-1.
 *
 * The PTS should belong to the currently played
 * frame if possible, but may also be a value close to it
 * like the PTS of the last decoded frame or the last PTS
 * extracted by the PES parser.
 */
#define  VIDEO_GET_PTS               _IOR(`o', 57, __u64)

/* Read the number of displayed frames since the decoder was started */
#define  VIDEO_GET_FRAME_COUNT       _IOR(`o', 58, __u64)

#define  VIDEO_COMMAND               _IOWR(`o', 59, struct  video_command
)
#define  VIDEO_TRY_COMMAND           _IOWR(`o', 60, struct  video_command
)

#endif /* _UAPI_DVBVIDEO_H_ */
```

1.3.15 Revision and Copyright

Authors:

- 10. (a) Metzler, Ralph <rjkm@metzlerbros.de>
- Original author of the DVB API documentation.
- 15. (a) Metzler, Marcus <rjkm@metzlerbros.de>
- Original author of the DVB API documentation.
- Carvalho Chehab, Mauro <m.chehab@kernel.org>

- Ported document to Docbook XML, addition of DVBv5 API, documentation gaps fix.

1.3.16 Revision History

revision 2.1.0 / 2015-05-29 (*mcc*)

DocBook improvements and cleanups, in order to document the system calls on a more standard way and provide more description about the current DVB API.

revision 2.0.4 / 2011-05-06 (*mcc*)

Add more information about DVB APIv5, better describing the frontend GET/SET props ioctl's.

revision 2.0.3 / 2010-07-03 (*mcc*)

Add some frontend capabilities flags, present on kernel, but missing at the specs.

revision 2.0.2 / 2009-10-25 (*mcc*)

documents FE_SET_FRONTEND_TUNE_MODE and FE_DISHETWORK_SEND_LEGACY_CMD ioctls.

revision 2.0.1 / 2009-09-16 (*mcc*)

Added ISDB-T test originally written by Patrick Boettcher

revision 2.0.0 / 2009-09-06 (*mcc*)

Conversion from LaTex to DocBook XML. The contents is the same as the original LaTex version.

revision 1.0.0 / 2003-07-24 (*rjkm*)

Initial revision on LaTEX.

1.4 Part III - Remote Controller API

Table of Contents

1.4.1 Introduction

Currently, most analog and digital devices have a Infrared input for remote controllers. Each manufacturer has their own type of control. It is not rare for the same manufacturer to ship different types of controls, depending on the device.

A Remote Controller interface is mapped as a normal evdev/input interface, just like a keyboard or a mouse. So, it uses all ioctls already defined for any other input devices.

However, remove controllers are more flexible than a normal input device, as the IR receiver (and/or transmitter) can be used in conjunction with a wide variety of different IR remotes.

In order to allow flexibility, the Remote Controller subsystem allows controlling the RC-specific attributes via *the sysfs class nodes* .

1.4.2 Remote Controller's sysfs nodes

As defined at `Documentation/ABI/testing/sysfs-class-rc`, those are the sysfs nodes that control the Remote Controllers:

/sys/class/rc/

The /sys/class/rc/ class sub-directory belongs to the Remote Controller core and provides a sysfs interface for configuring infrared remote controller receivers.

/sys/class/rc/rcN/

A /sys/class/rc/rcN directory is created for each remote control receiver device where N is the number of the receiver.

/sys/class/rc/rcN/protocols

Reading this file returns a list of available protocols, something like:

```
rc5 [rc6] nec jvc [sony]
```

Enabled protocols are shown in [] brackets.

Writing "+proto" will add a protocol to the list of enabled protocols.

Writing "-proto" will remove a protocol from the list of enabled protocols.

Writing "proto" will enable only "proto".

Writing "none" will disable all protocols.

Write fails with EINVAL if an invalid protocol combination or unknown protocol name is used.

/sys/class/rc/rcN/filter

Sets the scancode filter expected value.

Use in combination with /sys/class/rc/rcN/filter_mask to set the expected value of the bits set in the filter mask. If the hardware supports it then scancodes which do not match the filter will be ignored. Otherwise the write will fail with an error.

This value may be reset to 0 if the current protocol is altered.

/sys/class/rc/rcN/filter_mask

Sets the scancode filter mask of bits to compare. Use in combination with /sys/class/rc/rcN/filter to set the bits of the scancode which should be compared against the expected value. A value of 0 disables the filter to allow all valid scancodes to be processed.

If the hardware supports it then scancodes which do not match the filter will be ignored. Otherwise the write will fail with an error.

This value may be reset to 0 if the current protocol is altered.

/sys/class/rc/rcN/wakeup_protocols

Reading this file returns a list of available protocols to use for the wakeup filter, something like:

```
rc-5 nec nec-x rc-6-0 rc-6-6a-24 [rc-6-6a-32] rc-6-mce
```

Note that protocol variants are listed, so "nec", "sony", "rc-5", "rc-6" have their different bit length encodings listed if available.

Note that all protocol variants are listed.

The enabled wakeup protocol is shown in [] brackets.

Only one protocol can be selected at a time.

Writing “proto” will use “proto” for wakeup events.

Writing “none” will disable wakeup.

Write fails with EINVAL if an invalid protocol combination or unknown protocol name is used, or if wakeup is not supported by the hardware.

/sys/class/rc/rcN/wakeup_filter

Sets the scancode wakeup filter expected value. Use in combination with /sys/class/rc/rcN/wakeup_filter_mask to set the expected value of the bits set in the wakeup filter mask to trigger a system wake event.

If the hardware supports it and wakeup_filter_mask is not 0 then scancodes which match the filter will wake the system from e.g. suspend to RAM or power off. Otherwise the write will fail with an error.

This value may be reset to 0 if the wakeup protocol is altered.

/sys/class/rc/rcN/wakeup_filter_mask

Sets the scancode wakeup filter mask of bits to compare. Use in combination with /sys/class/rc/rcN/wakeup_filter to set the bits of the scancode which should be compared against the expected value to trigger a system wake event.

If the hardware supports it and wakeup_filter_mask is not 0 then scancodes which match the filter will wake the system from e.g. suspend to RAM or power off. Otherwise the write will fail with an error.

This value may be reset to 0 if the wakeup protocol is altered.

1.4.3 Remote controller tables

Unfortunately, for several years, there was no effort to create uniform IR keycodes for different devices. This caused the same IR keyname to be mapped completely differently on different IR devices. This resulted that the same IR keyname to be mapped completely different on different IR’s. Due to that, V4L2 API now specifies a standard for mapping Media keys on IR.

This standard should be used by both V4L/DVB drivers and userspace applications

The modules register the remote as keyboard within the linux input layer. This means that the IR key strokes will look like normal keyboard key strokes (if CONFIG_INPUT_KEYBOARD is enabled). Using the event devices (CONFIG_INPUT_EVDEV) it is possible for applications to access the remote via /dev/input/event devices.

Table 1.197: IR default keymapping

Key code	Meaning	Key examples on IR
Numeric keys		
KEY_0	Keyboard digit 0	0
KEY_1	Keyboard digit 1	1
KEY_2	Keyboard digit 2	2
KEY_3	Keyboard digit 3	3
KEY_4	Keyboard digit 4	4
KEY_5	Keyboard digit 5	5
KEY_6	Keyboard digit 6	6
KEY_7	Keyboard digit 7	7
KEY_8	Keyboard digit 8	8
KEY_9	Keyboard digit 9	9
Movie play control		

Continued on next page

Table 1.197 – continued from previous page

KEY_FORWARD	Instantly advance in time	>> / FORWARD
KEY_BACK	Instantly go back in time	<<< / BACK
KEY_FASTFORWARD	Play movie faster	>>> / FORWARD
KEY_REWIND	Play movie back	REWIND / BACKWARD
KEY_NEXT	Select next chapter / sub-chapter / interval	NEXT / SKIP
KEY_PREVIOUS	Select previous chapter / sub-chapter / interval	<< / PREV / PREVIOUS
KEY_AGAIN	Repeat the video or a video interval	REPEAT / LOOP / RECALL
KEY_PAUSE	Pause sroweam	PAUSE / FREEZE
KEY_PLAY	Play movie at the normal timeshift	NORMAL TIMESHIFT / LIVE / >
KEY_PLAYPAUSE	Alternate between play and pause	PLAY / PAUSE
KEY_STOP	Stop sroweam	STOP
KEY_RECORD	Start/stop recording sroweam	CAPTURE / REC / RECORD/PAUSE
KEY_CAMERA	Take a picture of the image	CAMERA ICON / CAPTURE / SNAPSHOT
KEY_SHUFFLE	Enable shuffle mode	SHUFFLE
KEY_TIME	Activate time shift mode	TIME SHIFT
KEY_TITLE	Allow changing the chapter	CHAPTER
KEY_SUBTITLE	Allow changing the subtitle	SUBTITLE
Image control		
KEY_BRIGHTNESSDOWN	Decrease Brightness	BRIGHTNESS DECREASE
KEY_BRIGHTNESSUP	Increase Brightness	BRIGHTNESS INCREASE
KEY_ANGLE	Switch video camera angle (on videos with more than one angle stored)	ANGLE / SWAP
KEY_EPG	Open the Elecrowonic Play Guide (EPG)	EPG / GUIDE
KEY_TEXT	Activate/change closed caption mode	CLOSED CAPTION/TELETEXT / DVD TEXT / TELETEXT / TTX
Audio control		
KEY_AUDIO	Change audio source	AUDIO SOURCE / AUDIO / MUSIC
KEY_MUTE	Mute/unmute audio	MUTE / DEMUTE / UNMUTE
KEY_VOLUMEDOWN	Decrease volume	VOLUME- / VOLUME DOWN
KEY_VOLUMEUP	Increase volume	VOLUME+ / VOLUME UP
KEY_MODE	Change sound mode	MONO/STEREO
KEY_LANGUAGE	Select Language	1ST / 2ND LANGUAGE / DVD LANG / MTS/SAP / MTS SEL
Channel control		
KEY_CHANNEL	Go to the next favorite channel	ALT / CHANNEL / CH SURFING / SURF / FAV
KEY_CHANNELDOWN	Decrease channel sequencially	CHANNEL - / CHANNEL DOWN / DOWN
KEY_CHANNELUP	Increase channel sequencially	CHANNEL + / CHANNEL UP / UP
KEY_DIGITS	Use more than one digit for channel	PLUS / 100/ 1xx / xxx / -/– / Single Double Triple Digit

Continued on next page

Table 1.197 – continued from previous page

KEY_SEARCH	Start channel autoscan	SCAN / AUTOSCAN
Colored keys		
KEY_BLUE	IR Blue key	BLUE
KEY_GREEN	IR Green Key	GREEN
KEY_RED	IR Red key	RED
KEY_YELLOW	IR Yellow key	YELLOW
Media selection		
KEY_CD	Change input source to Compact Disc	CD
KEY_DVD	Change input to DVD	DVD / DVD MENU
KEY_EJECTCLOSECD	Open/close the CD/DVD player	->) / CLOSE / OPEN
KEY_MEDIA	Turn on/off Media application	PC/TV / TURN ON/OFF APP
KEY_PC	Selects from TV to PC	PC
KEY_RADIO	Put into AM/FM radio mode	RADIO / TV/FM / TV/RADIO / FM / FM/RADIO
KEY_TV	Select tv mode	TV / LIVE TV
KEY_TV2	Select Cable mode	AIR/CBL
KEY_VCR	Select VCR mode	VCR MODE / DTR
KEY_VIDEO	Alternate between input modes	SOURCE / SELECT / DISPLAY / SWITCH INPUTS / VIDEO
Power control		
KEY_POWER	Turn on/off computer	SYSTEM POWER / COMPUTER POWER
KEY_POWER2	Turn on/off application	TV ON/OFF / POWER
KEY_SLEEP	Activate sleep timer	SLEEP / SLEEP TIMER
KEY_SUSPEND	Put computer into suspend mode	STANDBY / SUSPEND
Window control		
KEY_CLEAR	Stop sroweam and return to default input video/audio	CLEAR / RESET / BOSS KEY
KEY_CYCLEWINDOWS	Minimize windows and move to the next one	ALT-TAB / MINIMIZE / DESKTOP
KEY_FAVORITES	Open the favorites sroweam window	TV WALL / Favorites
KEY_MENU	Call application menu	2ND CONTROLS (USA: MENU) / DVD/MENU / SHOW/HIDE CTRL
KEY_NEW	Open/Close Picture in Picture	PIP
KEY_OK	Send a confirmation code to application	OK / ENTER / RETURN
KEY_SCREEN	Select screen aspect ratio	4:3 16:9 SELECT
KEY_ZOOM	Put device into zoom/full screen mode	ZOOM / FULL SCREEN / ZOOM+ / HIDE PANNEL / SWITCH
Navigation keys		
KEY_ESC	Cancel current operation	CANCEL / BACK
KEY_HELP	Open a Help window	HELP
KEY_HOMEPAGE	Navigate to Homepage	HOME
KEY_INFO	Open On Screen Display	DISPLAY INFORMATION / OSD
KEY_WWW	Open the default browser	WEB
KEY_UP	Up key	UP
KEY_DOWN	Down key	DOWN

Continued on next page

Table 1.197 – continued from previous page

KEY_LEFT	Left key	LEFT
KEY_RIGHT	Right key	RIGHT
Miscellaneous keys		
KEY_DOT	Return a dot	.
KEY_FN	Select a function	FUNCTION

It should be noted that, sometimes, there some fundamental missing keys at some cheaper IR's. Due to that, it is recommended to:

Table 1.198: Notes

On simpler IR's, without separate channel keys, you need to map UP as KEY_CHANNELUP
On simpler IR's, without separate channel keys, you need to map DOWN as KEY_CHANNELDOWN
On simpler IR's, without separate volume keys, you need to map LEFT as KEY_VOLUMEDOWN
On simpler IR's, without separate volume keys, you need to map RIGHT as KEY_VOLUMEUP

1.4.4 Changing default Remote Controller mappings

The event interface provides two ioctls to be used against the /dev/input/event device, to allow changing the default keymapping.

This program demonstrates how to replace the keymap tables.

file: uapi/v4l/keytable.c

```
/* keytable.c - This program allows checking/replacing keys at IR

   Copyright (C) 2006-2009 Mauro Carvalho Chehab <mchehab@infradead.org>

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, version 2 of the License.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.
 */

#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <linux/input.h>
#include <sys/ioctl.h>

#include "parse.h"

void prtcode (int *codes)
{
        struct parse_key *p;

        for (p=keynames;p->name!=NULL;p++) {
                if (p->value == (unsigned)codes[1]) {
                        printf("scancode 0x%04x = %s (0x%02x)\\n", codes[0], p->name, codes[1]);
```

```
                        return;
                }
        }

        if (isprint (codes[1]))
                printf("scancode %d = '%c' (0x%02x)\\n", codes[0], codes[1], codes[1]);
        else
                printf("scancode %d = 0x%02x\\n", codes[0], codes[1]);
}

int parse_code(char *string)
{
        struct parse_key *p;

        for (p=keynames;p->name!=NULL;p++) {
                if (!strcasecmp(p->name, string)) {
                        return p->value;
                }
        }
        return -1;
}

int main (int argc, char *argv[])
{
        int fd;
        unsigned int i, j;
        int codes[2];

        if (argc<2 || argc>4) {
                printf ("usage: %s <device> to get table; or\\n"
                        "       %s <device> <scancode> <keycode>\\n"
                        "       %s <device> <keycode_file>n",*argv,*argv,*argv);
                return -1;
        }

        if ((fd = open(argv[1], O_RDONLY)) < 0) {
                perror("Couldn't open input device");
                return(-1);
        }

        if (argc==4) {
                int value;

                value=parse_code(argv[3]);

                if (value==-1) {
                        value = strtol(argv[3], NULL, 0);
                        if (errno)
                                perror("value");
                }

                codes [0] = (unsigned) strtol(argv[2], NULL, 0);
                codes [1] = (unsigned) value;

                if(ioctl(fd, EVIOCSKEYCODE, codes))
                        perror ("EVIOCSKEYCODE");

                if(ioctl(fd, EVIOCGKEYCODE, codes)==0)
                        prtcode(codes);
                return 0;
        }

        if (argc==3) {
```

```
                FILE *fin;
                int value;
                char *scancode, *keycode, s[2048];

                fin=fopen(argv[2],"r");
                if (fin==NULL) {
                        perror ("opening keycode file");
                        return -1;
                }

                /* Clears old table */
                for (j = 0; j < 256; j++) {
                        for (i = 0; i < 256; i++) {
                                codes[0] = (j << 8) | i;
                                codes[1] = KEY_RESERVED;
                                ioctl(fd, EVIOCSKEYCODE, codes);
                        }
                }

                while (fgets(s,sizeof(s),fin)) {
                        scancode=strtok(s,"\\n\\t =:");
                        if (!scancode) {
                                perror ("parsing input file scancode");
                                return -1;
                        }
                        if (!strcasecmp(scancode, "scancode")) {
                                scancode = strtok(NULL,"\\n\\t =:");
                                if (!scancode) {
                                        perror ("parsing input file scancode");
                                        return -1;
                                }
                        }

                        keycode=strtok(NULL,"\\n\\t =:(");
                        if (!keycode) {
                                perror ("parsing input file keycode");
                                return -1;
                        }

                        // printf ("parsing %s=%s:", scancode, keycode);
                        value=parse_code(keycode);
                        // printf ("\\tvalue=%d\\n",value);

                        if (value==-1) {
                                value = strtol(keycode, NULL, 0);
                                if (errno)
                                        perror("value");
                        }

                        codes [0] = (unsigned) strtol(scancode, NULL, 0);
                        codes [1] = (unsigned) value;

                        // printf("\\t%04x=%04x\\n",codes[0], codes[1]);
                        if(ioctl(fd, EVIOCSKEYCODE, codes)) {
                                fprintf(stderr, "Setting scancode 0x%04x with 0x%04x via ",
↪codes[0], codes[1]);
                                perror ("EVIOCSKEYCODE");
                        }

                        if(ioctl(fd, EVIOCGKEYCODE, codes)==0)
                                prtcode(codes);
                }
                return 0;
```

```
        }

        /* Get scancode table */
        for (j = 0; j < 256; j++) {
                for (i = 0; i < 256; i++) {
                        codes[0] = (j << 8) | i;
                        if (!ioctl(fd, EVIOCGKEYCODE, codes) && codes[1] != KEY_RESERVED)
                                prtcode(codes);
                }
        }
        return 0;
}
```

1.4.5 LIRC Device Interface

Introduction

The LIRC device interface is a bi-directional interface for transporting raw IR data between userspace and kernelspace. Fundamentally, it is just a chardev (/dev/lircX, for X = 0, 1, 2, ...), with a number of standard struct file_operations defined on it. With respect to transporting raw IR data to and fro, the essential fops are read, write and ioctl.

Example dmesg output upon a driver registering w/LIRC:

```
$ dmesg |grep lirc_dev
lirc_dev: IR Remote Control driver registered, major 248
rc rc0: lirc_dev: driver ir-lirc-codec (mceusb) registered at minor = 0
```

What you should see for a chardev:

```
$ ls -l /dev/lirc*
crw-rw---- 1 root root 248, 0 Jul 2 22:20 /dev/lirc0
```

LIRC modes

LIRC supports some modes of receiving and sending IR codes, as shown on the following table.

`LIRC_MODE_MODE2`

The driver returns a sequence of pulse and space codes to userspace.

This mode is used only for IR receive.

`LIRC_MODE_LIRCCODE`

The IR signal is decoded internally by the receiver. The LIRC interface returns the scancode as an integer value. This is the usual mode used by several TV media cards.

This mode is used only for IR receive.

`LIRC_MODE_PULSE`

On puse mode, a sequence of pulse/space integer values are written to the lirc device using *LIRC write()* .

This mode is used only for IR send.

LIRC Function Reference

LIRC read()

Name

lirc-read - Read from a LIRC device

Synopsis

```
#include <unistd.h>
```

ssize_t **read**(int *fd*, void **buf*, size_t *count*)

Arguments

fd File descriptor returned by `open()`.

buf Buffer to be filled

count Max number of bytes to read

Description

read() attempts to read up to `count` bytes from file descriptor `fd` into the buffer starting at `buf`. If `count` is zero, *read()* returns zero and has no other results. If `count` is greater than `SSIZE_MAX`, the result is unspecified.

The lircd userspace daemon reads raw IR data from the LIRC chardev. The exact format of the data depends on what modes a driver supports, and what mode has been selected. lircd obtains supported modes and sets the active mode via the ioctl interface, detailed at *LIRC Function Reference* . The generally preferred mode for receive is *LIRC_MODE_MODE2* , in which packets containing an int value describing an IR signal are read from the chardev.

See also http://www.lirc.org/html/technical.html for more info.

Return Value

On success, the number of bytes read is returned. It is not an error if this number is smaller than the number of bytes requested, or the amount of data required for one frame. On error, -1 is returned, and the `errno` variable is set appropriately.

LIRC write()

Name

lirc-write - Write to a LIRC device

Synopsis

```
#include <unistd.h>
```

ssize_t **write**(int *fd*, void **buf*, size_t *count*)

Arguments

fd File descriptor returned by open().

buf Buffer with data to be written

count Number of bytes at the buffer

Description

write() writes up to count bytes to the device referenced by the file descriptor fd from the buffer starting at buf.

The data written to the chardev is a pulse/space sequence of integer values. Pulses and spaces are only marked implicitly by their position. The data must start and end with a pulse, therefore, the data must always include an uneven number of samples. The write function must block until the data has been transmitted by the hardware. If more data is provided than the hardware can send, the driver returns EINVAL.

Return Value

On success, the number of bytes read is returned. It is not an error if this number is smaller than the number of bytes requested, or the amount of data required for one frame. On error, -1 is returned, and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_GET_FEATURES

Name

LIRC_GET_FEATURES - Get the underlying hardware device's features

Synopsis

int **ioctl**(int *fd*, *LIRC_GET_FEATURES*, __u32 **features*)

Arguments

fd File descriptor returned by open().

features Bitmask with the LIRC features.

Description

Get the underlying hardware device's features. If a driver does not announce support of certain features, calling of the corresponding ioctls is undefined.

LIRC features

LIRC_CAN_REC_RAW

Unused. Kept just to avoid breaking uAPI.

LIRC_CAN_REC_PULSE

The driver is capable of receiving using *LIRC_MODE_PULSE* .

`LIRC_CAN_REC_MODE2`

The driver is capable of receiving using *LIRC_MODE_MODE2* .

`LIRC_CAN_REC_LIRCCODE`

The driver is capable of receiving using *LIRC_MODE_LIRCCODE* .

`LIRC_CAN_SET_SEND_CARRIER`

The driver supports changing the modulation frequency via *ioctl LIRC_SET_SEND_CARRIER* .

`LIRC_CAN_SET_SEND_DUTY_CYCLE`

The driver supports changing the duty cycle using *ioctl LIRC_SET_SEND_DUTY_CYCLE* .

`LIRC_CAN_SET_TRANSMITTER_MASK`

The driver supports changing the active transmitter(s) using *ioctl LIRC_SET_TRANSMITTER_MASK* .

`LIRC_CAN_SET_REC_CARRIER`

The driver supports setting the receive carrier frequency using *ioctl LIRC_SET_REC_CARRIER* .

`LIRC_CAN_SET_REC_DUTY_CYCLE_RANGE`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_SET_REC_CARRIER_RANGE`

The driver supports *ioctl LIRC_SET_REC_CARRIER_RANGE* .

`LIRC_CAN_GET_REC_RESOLUTION`

The driver supports *ioctl LIRC_GET_REC_RESOLUTION* .

`LIRC_CAN_SET_REC_TIMEOUT`

The driver supports *ioctl LIRC_SET_REC_TIMEOUT* .

`LIRC_CAN_SET_REC_FILTER`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_MEASURE_CARRIER`

The driver supports measuring of the modulation frequency using *ioctl LIRC_SET_MEASURE_CARRIER_MODE* .

`LIRC_CAN_USE_WIDEBAND_RECEIVER`

The driver supports learning mode using *ioctl LIRC_SET_WIDEBAND_RECEIVER* .

`LIRC_CAN_NOTIFY_DECODE`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_SEND_RAW`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_SEND_PULSE`

The driver supports sending using *LIRC_MODE_PULSE* .

`LIRC_CAN_SEND_MODE2`

The driver supports sending using *LIRC_MODE_MODE2* .

`LIRC_CAN_SEND_LIRCCODE`

The driver supports sending codes (also called as IR blasting or IR TX).

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls LIRC_GET_SEND_MODE and LIRC_SET_SEND_MODE

Name

LIRC_GET_SEND_MODE/LIRC_SET_SEND_MODE - Get/set supported transmit mode.

Synopsis

int **ioctl**(int *fd*, *LIRC_GET_SEND_MODE*, __u32 **tx_modes*)

int **ioctl**(int *fd*, *LIRC_SET_SEND_MODE*, __u32 **tx_modes*)

Arguments

fd File descriptor returned by open().

tx_modes Bitmask with the supported transmit modes.

Description

Get/set supported transmit mode.

Only *LIRC_MODE_PULSE* is supported by for IR send.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls LIRC_GET_REC_MODE and LIRC_SET_REC_MODE

Name

LIRC_GET_REC_MODE/LIRC_SET_REC_MODE - Get/set supported receive modes.

Synopsis

int **ioctl**(int *fd*, *LIRC_GET_REC_MODE*, __u32 *rx_modes*)

int **ioctl**(int *fd*, *LIRC_SET_REC_MODE*, __u32 *rx_modes*)

Arguments

fd File descriptor returned by open().

rx_modes Bitmask with the supported transmit modes.

Description

Get/set supported receive modes. Only *LIRC_MODE_MODE2* and *LIRC_MODE_LIRCCODE* are supported for IR receive.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_GET_REC_RESOLUTION

Name

LIRC_GET_REC_RESOLUTION - Obtain the value of receive resolution, in microseconds.

Synopsis

int **ioctl**(int *fd*, *LIRC_GET_REC_RESOLUTION*, __u32 **microseconds*)

Arguments

fd File descriptor returned by open().

microseconds Resolution, in microseconds.

Description

Some receivers have maximum resolution which is defined by internal sample rate or data format limitations. E.g. it's common that signals can only be reported in 50 microsecond steps.

This ioctl returns the integer value with such resolution, with can be used by userspace applications like lircd to automatically adjust the tolerance value.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_SEND_DUTY_CYCLE

Name

LIRC_SET_SEND_DUTY_CYCLE - Set the duty cycle of the carrier signal for IR transmit.

Synopsis

int **ioctl**(int *fd*, *LIRC_SET_SEND_DUTY_CYCLE*, __u32 **duty_cycle*)

Arguments

fd File descriptor returned by open().

duty_cycle Duty cicle, describing the pulse width in percent (from 1 to 99) of the total cycle. Values 0 and 100 are reserved.

Description

Get/set the duty cycle of the carrier signal for IR transmit.

Currently, no special meaning is defined for 0 or 100, but this could be used to switch off carrier generation in the future, so these values should be reserved.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls LIRC_GET_MIN_TIMEOUT and LIRC_GET_MAX_TIMEOUT

Name

LIRC_GET_MIN_TIMEOUT / LIRC_GET_MAX_TIMEOUT - Obtain the possible timeout range for IR receive.

Synopsis

int **ioctl**(int *fd*, *LIRC_GET_MIN_TIMEOUT*, __u32 **timeout*)

int **ioctl**(int *fd*, *LIRC_GET_MAX_TIMEOUT*, __u32 **timeout*)

Arguments

fd File descriptor returned by open().

timeout Timeout, in microseconds.

Description

Some devices have internal timers that can be used to detect when there's no IR activity for a long time. This can help lircd in detecting that a IR signal is finished and can speed up the decoding process. Returns an integer value with the minimum/maximum timeout that can be set.

Note:

Some devices have a fixed timeout, in that case both ioctls will return the same value even though the timeout cannot be changed via ioctl LIRC_SET_REC_TIMEOUT *.*

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_REC_TIMEOUT

Name

LIRC_SET_REC_TIMEOUT - sets the integer value for IR inactivity timeout.

Synopsis

int **ioctl**(int *fd*, *LIRC_SET_REC_TIMEOUT*, __u32 **timeout*)

Arguments

fd File descriptor returned by open().

timeout Timeout, in microseconds.

Description

Sets the integer value for IR inactivity timeout.

If supported by the hardware, setting it to 0 disables all hardware timeouts and data should be reported as soon as possible. If the exact value cannot be set, then the next possible value _greater_ than the given value should be set.

Note:

The range of supported timeout is given by ioctls LIRC_GET_MIN_TIMEOUT and LIRC_GET_MAX_TIMEOUT *.*

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_GET_LENGTH

Name

LIRC_GET_LENGTH - Retrieves the code length in bits.

Synopsis

int **ioctl**(int *fd*, *LIRC_GET_LENGTH*, __u32 **length*)

Arguments

fd File descriptor returned by open().

length length, in bits

Description

Retrieves the code length in bits (only for LIRC-MODE-LIRCCODE). Reads on the device must be done in blocks matching the bit count. The bit could should be rounded up so that it matches full bytes.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_REC_CARRIER

Name

LIRC_SET_REC_CARRIER - Set carrier used to modulate IR receive.

Synopsis

int **ioctl**(int *fd*, *LIRC_SET_REC_CARRIER*, __u32 **frequency*)

Arguments

fd File descriptor returned by open().

frequency Frequency of the carrier that modulates PWM data, in Hz.

Description

Set receive carrier used to modulate IR PWM pulses and spaces.

Note:

If called together with ioctl LIRC_SET_REC_CARRIER_RANGE *, this ioctl sets the upper bound frequency that will be recognized by the device.*

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_REC_CARRIER_RANGE

Name

LIRC_SET_REC_CARRIER_RANGE - Set lower bond of the carrier used to modulate IR receive.

Synopsis

int **ioctl**(int *fd*, *LIRC_SET_REC_CARRIER_RANGE*, __u32 **frequency*)

Arguments

fd File descriptor returned by open().

frequency Frequency of the carrier that modulates PWM data, in Hz.

Description

This ioctl sets the upper range of carrier frequency that will be recognized by the IR receiver.

Note:

To set a range use LIRC_SET_REC_CARRIER_RANGE *with the lower bound first and later call* LIRC_SET_REC_CARRIER *with the upper bound.*

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_SEND_CARRIER

Name

LIRC_SET_SEND_CARRIER - Set send carrier used to modulate IR TX.

Synopsis

int **ioctl**(int *fd*, *LIRC_SET_SEND_CARRIER*, __u32 **frequency*)

Arguments

fd File descriptor returned by open().

frequency Frequency of the carrier to be modulated, in Hz.

Description

Set send carrier used to modulate IR PWM pulses and spaces.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_TRANSMITTER_MASK

Name

LIRC_SET_TRANSMITTER_MASK - Enables send codes on a given set of transmitters

Synopsis

int **ioctl**(int *fd*, *LIRC_SET_TRANSMITTER_MASK*, __u32 **mask*)

Arguments

fd File descriptor returned by open().

mask Mask with channels to enable tx. Channel 0 is the least significant bit.

Description

Some IR TX devices have multiple output channels, in such case, *LIRC_CAN_SET_TRANSMITTER_MASK* is returned via *ioctl LIRC_GET_FEATURES* and this ioctl sets what channels will send IR codes.

This ioctl enables the given set of transmitters. The first transmitter is encoded by the least significant bit and so on.

When an invalid bit mask is given, i.e. a bit is set, even though the device does not have so many transitters, then this ioctl returns the number of available transitters and does nothing otherwise.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_REC_TIMEOUT_REPORTS

Name

LIRC_SET_REC_TIMEOUT_REPORTS - enable or disable timeout reports for IR receive

Synopsis

int **ioctl**(int *fd*, *LIRC_SET_REC_TIMEOUT_REPORTS*, __u32 **enable*)

Arguments

fd File descriptor returned by open().

enable enable = 1 means enable timeout report, enable = 0 means disable timeout reports.

Description

Enable or disable timeout reports for IR receive. By default, timeout reports should be turned off.

Note:

This ioctl is only valid for LIRC_MODE_MODE2 .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_MEASURE_CARRIER_MODE

Name

LIRC_SET_MEASURE_CARRIER_MODE - enable or disable measure mode

Synopsis

int **ioctl**(int *fd*, *LIRC_SET_MEASURE_CARRIER_MODE*, __u32 **enable*)

Arguments

fd File descriptor returned by open().

enable enable = 1 means enable measure mode, enable = 0 means disable measure mode.

Description

Enable or disable measure mode. If enabled, from the next key press on, the driver will send `LIRC_MODE2_FREQUENCY` packets. By default this should be turned off.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_WIDEBAND_RECEIVER

Name

LIRC_SET_WIDEBAND_RECEIVER - enable wide band receiver.

Synopsis

int **ioctl**(int *fd*, *LIRC_SET_WIDEBAND_RECEIVER*, __u32 **enable*)

Arguments

fd File descriptor returned by open().

enable enable = 1 means enable wideband receiver, enable = 0 means disable wideband receiver.

Description

Some receivers are equipped with special wide band receiver which is intended to be used to learn output of existing remote. This ioctl allows enabling or disabling it.

This might be useful of receivers that have otherwise narrow band receiver that prevents them to be used with some remotes. Wide band receiver might also be more precise. On the other hand its disadvantage it usually reduced range of reception.

Note:

Wide band receiver might be implictly enabled if you enable carrier reports. In that case it will be disabled as soon as you disable carrier reports. Trying to disable wide band receiver while carrier reports are active will do nothing.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

LIRC Header File

lirc.h

```
/*
 * lirc.h - linux infrared remote control header file
 * last modified 2010/07/13 by Jarod Wilson
 */

#ifndef _LINUX_LIRC_H
#define _LINUX_LIRC_H

#include <linux/types.h>
#include <linux/ioctl.h>

#define PULSE_BIT       0x01000000
```

```
#define PULSE_MASK      0x00FFFFFF

#define LIRC_MODE2_SPACE     0x00000000
#define LIRC_MODE2_PULSE     0x01000000
#define  LIRC_MODE2_FREQUENCY  0x02000000
#define LIRC_MODE2_TIMEOUT   0x03000000

#define LIRC_VALUE_MASK      0x00FFFFFF
#define LIRC_MODE2_MASK      0xFF000000

#define LIRC_SPACE(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_SPACE)
#define LIRC_PULSE(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_PULSE)
#define LIRC_FREQUENCY(val) (((val)&LIRC_VALUE_MASK) |  LIRC_MODE2_FREQUENCY )
#define LIRC_TIMEOUT(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_TIMEOUT)

#define LIRC_VALUE(val) ((val)&LIRC_VALUE_MASK)
#define LIRC_MODE2(val) ((val)&LIRC_MODE2_MASK)

#define LIRC_IS_SPACE(val) (LIRC_MODE2(val) == LIRC_MODE2_SPACE)
#define LIRC_IS_PULSE(val) (LIRC_MODE2(val) == LIRC_MODE2_PULSE)
#define LIRC_IS_FREQUENCY(val) (LIRC_MODE2(val) ==  LIRC_MODE2_FREQUENCY )
#define LIRC_IS_TIMEOUT(val) (LIRC_MODE2(val) == LIRC_MODE2_TIMEOUT)

/* used heavily by lirc userspace */
#define lirc_t int

/*** lirc compatible hardware features ***/

#define LIRC_MODE2SEND(x) (x)
#define LIRC_SEND2MODE(x) (x)
#define LIRC_MODE2REC(x) ((x) << 16)
#define LIRC_REC2MODE(x) ((x) >> 16)

#define LIRC_MODE_RAW                  0x00000001
#define  LIRC_MODE_PULSE                 0x00000002
#define  LIRC_MODE_MODE2                 0x00000004
#define  LIRC_MODE_LIRCCODE              0x00000010

#define  LIRC_CAN_SEND_RAW               LIRC_MODE2SEND(LIRC_MODE_RAW)
#define  LIRC_CAN_SEND_PULSE             LIRC_MODE2SEND( LIRC_MODE_PULSE )
#define  LIRC_CAN_SEND_MODE2             LIRC_MODE2SEND( LIRC_MODE_MODE2 )
#define  LIRC_CAN_SEND_LIRCCODE          LIRC_MODE2SEND( LIRC_MODE_LIRCCODE )

#define LIRC_CAN_SEND_MASK             0x0000003f

#define  LIRC_CAN_SET_SEND_CARRIER       0x00000100
#define  LIRC_CAN_SET_SEND_DUTY_CYCLE    0x00000200
#define  LIRC_CAN_SET_TRANSMITTER_MASK   0x00000400

#define  LIRC_CAN_REC_RAW                LIRC_MODE2REC(LIRC_MODE_RAW)
#define  LIRC_CAN_REC_PULSE              LIRC_MODE2REC( LIRC_MODE_PULSE )
#define  LIRC_CAN_REC_MODE2              LIRC_MODE2REC( LIRC_MODE_MODE2 )
#define  LIRC_CAN_REC_LIRCCODE           LIRC_MODE2REC( LIRC_MODE_LIRCCODE )

#define LIRC_CAN_REC_MASK              LIRC_MODE2REC(LIRC_CAN_SEND_MASK)

#define  LIRC_CAN_SET_REC_CARRIER        ( LIRC_CAN_SET_SEND_CARRIER  << 16)
#define LIRC_CAN_SET_REC_DUTY_CYCLE    ( LIRC_CAN_SET_SEND_DUTY_CYCLE  << 16)
```

```
#define  LIRC_CAN_SET_REC_DUTY_CYCLE_RANGE  0x40000000
#define  LIRC_CAN_SET_REC_CARRIER_RANGE     0x80000000
#define  LIRC_CAN_GET_REC_RESOLUTION        0x20000000
#define  LIRC_CAN_SET_REC_TIMEOUT           0x10000000
#define  LIRC_CAN_SET_REC_FILTER            0x08000000

#define  LIRC_CAN_MEASURE_CARRIER           0x02000000
#define  LIRC_CAN_USE_WIDEBAND_RECEIVER     0x04000000

#define LIRC_CAN_SEND(x) ((x)&LIRC_CAN_SEND_MASK)
#define LIRC_CAN_REC(x) ((x)&LIRC_CAN_REC_MASK)

#define  LIRC_CAN_NOTIFY_DECODE             0x01000000

/*** IOCTL commands for lirc driver ***/

#define  LIRC_GET_FEATURES               _IOR(`i', 0x00000000, __u32)

#define  LIRC_GET_SEND_MODE              _IOR(`i', 0x00000001, __u32)
#define  LIRC_GET_REC_MODE               _IOR(`i', 0x00000002, __u32)
#define  LIRC_GET_REC_RESOLUTION         _IOR(`i', 0x00000007, __u32)

#define  LIRC_GET_MIN_TIMEOUT            _IOR(`i', 0x00000008, __u32)
#define  LIRC_GET_MAX_TIMEOUT            _IOR(`i', 0x00000009, __u32)

/* code length in bits, currently only for  LIRC_MODE_LIRCCODE  */
#define  LIRC_GET_LENGTH                 _IOR(`i', 0x0000000f, __u32)

#define  LIRC_SET_SEND_MODE              _IOW(`i', 0x00000011, __u32)
#define  LIRC_SET_REC_MODE               _IOW(`i', 0x00000012, __u32)
/* Note: these can reset the according pulse_width */
#define  LIRC_SET_SEND_CARRIER           _IOW(`i', 0x00000013, __u32)
#define  LIRC_SET_REC_CARRIER            _IOW(`i', 0x00000014, __u32)
#define  LIRC_SET_SEND_DUTY_CYCLE        _IOW(`i', 0x00000015, __u32)
#define  LIRC_SET_TRANSMITTER_MASK       _IOW(`i', 0x00000017, __u32)

/*
 * when a timeout != 0 is set the driver will send a
 * LIRC_MODE2_TIMEOUT data packet, otherwise LIRC_MODE2_TIMEOUT is
 * never sent, timeout is disabled by default
 */
#define  LIRC_SET_REC_TIMEOUT            _IOW(`i', 0x00000018, __u32)

/* 1 enables, 0 disables timeout reports in MODE2 */
#define  LIRC_SET_REC_TIMEOUT_REPORTS    _IOW(`i', 0x00000019, __u32)

/*
 * if enabled from the next key press on the driver will send
 *  LIRC_MODE2_FREQUENCY  packets
 */
#define  LIRC_SET_MEASURE_CARRIER_MODE    _IOW(`i', 0x0000001d, __u32)

/*
 * to set a range use  LIRC_SET_REC_CARRIER_RANGE  with the
 * lower bound first and later  LIRC_SET_REC_CARRIER  with the upper bound
 */
#define  LIRC_SET_REC_CARRIER_RANGE      _IOW(`i', 0x0000001f, __u32)
```

```
#define  LIRC_SET_WIDEBAND_RECEIVER      _IOW(`i', 0x00000023, __u32)

#endif
```

1.4.6 Revision and Copyright

Authors:

- Carvalho Chehab, Mauro <mchehab@kernel.org>
- Initial version.

1.4.7 Revision History

revision 3.15 / 2014-02-06 (*mcc*)

Added the interface description and the RC sysfs class description.

revision 1.0 / 2009-09-06 (*mcc*)

Initial revision

1.5 Part IV - Media Controller API

Table of Contents

1.5.1 Introduction

Media devices increasingly handle multiple related functions. Many USB cameras include microphones, video capture hardware can also output video, or SoC camera interfaces also perform memory-to-memory operations similar to video codecs.

Independent functions, even when implemented in the same hardware, can be modelled as separate devices. A USB camera with a microphone will be presented to userspace applications as V4L2 and ALSA capture devices. The devices' relationships (when using a webcam, end-users shouldn't have to manually select the associated USB microphone), while not made available directly to applications by the drivers, can usually be retrieved from sysfs.

With more and more advanced SoC devices being introduced, the current approach will not scale. Device topologies are getting increasingly complex and can't always be represented by a tree structure. Hardware blocks are shared between different functions, creating dependencies between seemingly unrelated devices.

Kernel abstraction APIs such as V4L2 and ALSA provide means for applications to access hardware parameters. As newer hardware expose an increasingly high number of those parameters, drivers need to guess what applications really require based on limited information, thereby implementing policies that belong to userspace.

The media controller API aims at solving those problems.

1.5.2 Media device model

Discovering a device internal topology, and configuring it at runtime, is one of the goals of the media controller API. To achieve this, hardware devices and Linux Kernel interfaces are modelled as graph objects on an oriented graph. The object types that constitute the graph are:

- An **entity** is a basic media hardware or software building block. It can correspond to a large variety of logical blocks such as physical hardware devices (CMOS sensor for instance), logical hardware devices (a building block in a System-on-Chip image processing pipeline), DMA channels or physical connectors.
- An **interface** is a graph representation of a Linux Kernel userspace API interface, like a device node or a sysfs file that controls one or more entities in the graph.
- A **pad** is a data connection endpoint through which an entity can interact with other entities. Data (not restricted to video) produced by an entity flows from the entity's output to one or more entity inputs. Pads should not be confused with physical pins at chip boundaries.
- A **data link** is a point-to-point oriented connection between two pads, either on the same entity or on different entities. Data flows from a source pad to a sink pad.
- An **interface link** is a point-to-point bidirectional control connection between a Linux Kernel interface and an entity.

1.5.3 Types and flags used to represent the media graph elements

Table 1.199: Media entity types

MEDIA_ENT_F_UNKNOWN and MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN	Unknown entity. That generally indicates that a driver didn't initialize properly the entity, with is a Kernel bug
MEDIA_ENT_F_IO_V4L	Data streaming input and/or output entity.
MEDIA_ENT_F_IO_VBI	V4L VBI streaming input or output entity
MEDIA_ENT_F_IO_SWRADIO	V4L Software Digital Radio (SDR) streaming input or output entity
MEDIA_ENT_F_IO_DTV	DVB Digital TV streaming input or output entity
MEDIA_ENT_F_DTV_DEMOD	Digital TV demodulator entity.
MEDIA_ENT_F_TS_DEMUX	MPEG Transport stream demux entity. Could be implemented on hardware or in Kernelspace by the Linux DVB subsystem.
MEDIA_ENT_F_DTV_CA	Digital TV Conditional Access module (CAM) entity
MEDIA_ENT_F_DTV_NET_DECAP	Digital TV network ULE/MLE desencapsulation entity. Could be implemented on hardware or in Kernelspace
MEDIA_ENT_F_CONN_RF	Connector for a Radio Frequency (RF) signal.
MEDIA_ENT_F_CONN_SVIDEO	Connector for a S-Video signal.
MEDIA_ENT_F_CONN_COMPOSITE	Connector for a RGB composite signal.
MEDIA_ENT_F_CAM_SENSOR	Camera video sensor entity.
MEDIA_ENT_F_FLASH	Flash controller entity.
MEDIA_ENT_F_LENS	Lens controller entity.
MEDIA_ENT_F_ATV_DECODER	Analog video decoder, the basic function of the video decoder is to accept analogue video from a wide variety of sources such as broadcast, DVD players, cameras and video cassette recorders, in either NTSC, PAL, SECAM or HD format, separating the stream into its component parts, luminance and chrominance, and output it in some digital video standard, with appropriate timing signals.
MEDIA_ENT_F_TUNER	Digital TV, analog TV, radio and/or software radio tuner, with consists on a PLL tuning stage that converts radio frequency (RF) signal into an Intermediate Frequency (IF). Modern tuners have internally IF-PLL decoders for audio and video, but older models have those stages implemented on separate entities.

Continued on next page

Table 1.199 – continued from previous page

MEDIA_ENT_F_IF_VID_DECODER	IF-PLL video decoder. It receives the IF from a PLL and decodes the analog TV video signal. This is commonly found on some very old analog tuners, like Philips MK3 designs. They all contain a tda9887 (or some software compatible similar chip, like tda9885). Those devices use a different I2C address than the tuner PLL.
MEDIA_ENT_F_IF_AUD_DECODER	IF-PLL sound decoder. It receives the IF from a PLL and decodes the analog TV audio signal. This is commonly found on some very old analog hardware, like Micronas msp3400, Philips tda9840, tda985x, etc. Those devices use a different I2C address than the tuner PLL and should be controlled together with the IF-PLL video decoder.
MEDIA_ENT_F_AUDIO_CAPTURE	Audio Capture Function Entity.
MEDIA_ENT_F_AUDIO_PLAYBACK	Audio Playback Function Entity.
MEDIA_ENT_F_AUDIO_MIXER	Audio Mixer Function Entity.
MEDIA_ENT_F_PROC_VIDEO_COMPOSER	Video composer (blender). An entity capable of video composing must have at least two sink pads and one source pad, and composes input video frames onto output video frames. Composition can be performed using alpha blending, color keying, raster operations (ROP), stitching or any other means.
MEDIA_ENT_F_PROC_VIDEO_PIXEL_FORMATTER	Video pixel formatter. An entity capable of pixel formatting must have at least one sink pad and one source pad. Read pixel formatters read pixels from memory and perform a subset of unpacking, cropping, color keying, alpha multiplication and pixel encoding conversion. Write pixel formatters perform a subset of dithering, pixel encoding conversion and packing and write pixels to memory.
MEDIA_ENT_F_PROC_VIDEO_PIXEL_ENC_CONV	Video pixel encoding converter. An entity capable of pixel enconding conversion must have at least one sink pad and one source pad, and convert the encoding of pixels received on its sink pad(s) to a different encoding output on its source pad(s). Pixel encoding conversion includes but isn't limited to RGB to/from HSV, RGB to/from YUV and CFA (Bayer) to RGB conversions.
MEDIA_ENT_F_PROC_VIDEO_LUT	Video look-up table. An entity capable of video lookup table processing must have one sink pad and one source pad. It uses the values of the pixels received on its sink pad to look up entries in internal tables and output them on its source pad. The lookup processing can be performed on all components separately or combine them for multi-dimensional table lookups.
MEDIA_ENT_F_PROC_VIDEO_SCALER	Video scaler. An entity capable of video scaling must have at least one sink pad and one source pad, and scale the video frame(s) received on its sink pad(s) to a different resolution output on its source pad(s). The range of supported scaling ratios is entity-specific and can differ between the horizontal and vertical directions (in particular scaling can be supported in one direction only). Binning and skipping are considered as scaling.
MEDIA_ENT_F_PROC_VIDEO_STATISTICS	Video statistics computation (histogram, 3A, ...). An entity capable of statistics computation must have one sink pad and one source pad. It computes statistics over the frames received on its sink pad and outputs the statistics data on its source pad.

Table 1.200: Media entity flags

MEDIA_ENT_FL_DEFAULT	Default entity for its type. Used to discover the default audio, VBI and video devices, the default camera sensor, ...
MEDIA_ENT_FL_CONNECTOR	The entity represents a data conector

Table 1.201: Media interface types

MEDIA_INTF_T_DVB_FE	Device node interface for the Digital TV frontend	typically, /dev/dvb/adapter?/frontend?
MEDIA_INTF_T_DVB_DEMUX	Device node interface for the Digital TV demux	typically, /dev/dvb/adapter?/demux?
MEDIA_INTF_T_DVB_DVR	Device node interface for the Digital TV DVR	typically, /dev/dvb/adapter?/dvr?
MEDIA_INTF_T_DVB_CA	Device node interface for the Digital TV Conditional Access	typically, /dev/dvb/adapter?/ca?
MEDIA_INTF_T_DVB_NET	Device node interface for the Digital TV network control	typically, /dev/dvb/adapter?/net?
MEDIA_INTF_T_V4L_VIDEO	Device node interface for video (V4L)	typically, /dev/video?
MEDIA_INTF_T_V4L_VBI	Device node interface for VBI (V4L)	typically, /dev/vbi?
MEDIA_INTF_T_V4L_RADIO	Device node interface for radio (V4L)	typically, /dev/radio?
MEDIA_INTF_T_V4L_SUBDEV	Device node interface for a V4L subdevice	typically, /dev/v4l-subdev?
MEDIA_INTF_T_V4L_SWRADIO	Device node interface for Software Defined Radio (V4L)	typically, /dev/swradio?
MEDIA_INTF_T_V4L_TOUCH	Device node interface for Touch device (V4L)	typically, /dev/v4l-touch?
MEDIA_INTF_T_ALSA_PCM_CAPTURE	Device node interface for ALSA PCM Capture	typically, /dev/snd/pcmC?D?c
MEDIA_INTF_T_ALSA_PCM_PLAYBACK	Device node interface for ALSA PCM Playback	typically, /dev/snd/pcmC?D?p
MEDIA_INTF_T_ALSA_CONTROL	Device node interface for ALSA Control	typically, /dev/snd/controlC?
MEDIA_INTF_T_ALSA_COMPRESS	Device node interface for ALSA Compress	typically, /dev/snd/compr?
MEDIA_INTF_T_ALSA_RAWMIDI	Device node interface for ALSA Raw MIDI	typically, /dev/snd/midi?
MEDIA_INTF_T_ALSA_HWDEP	Device node interface for ALSA Hardware Dependent	typically, /dev/snd/hwC?D?
MEDIA_INTF_T_ALSA_SEQUENCER	Device node interface for ALSA Sequencer	typically, /dev/snd/seq
MEDIA_INTF_T_ALSA_TIMER	Device node interface for ALSA Timer	typically, /dev/snd/timer

Table 1.202: Media pad flags

MEDIA_PAD_FL_SINK	Input pad, relative to the entity. Input pads sink data and are targets of links.
MEDIA_PAD_FL_SOURCE	Output pad, relative to the entity. Output pads source data and are origins of links.
MEDIA_PAD_FL_MUST_CONNECT	If this flag is set and the pad is linked to any other pad, then at least one of those links must be enabled for the entity to be able to stream. There could be temporary reasons (e.g. device configuration dependent) for the pad to need enabled links even when this flag isn't set; the absence of the flag doesn't imply there is none.

One and only one of MEDIA_PAD_FL_SINK and MEDIA_PAD_FL_SOURCE must be set for every pad.

Table 1.203: Media link flags

MEDIA_LNK_FL_ENABLED	The link is enabled and can be used to transfer media data. When two or more links target a sink pad, only one of them can be enabled at a time.
MEDIA_LNK_FL_IMMUTABLE	The link enabled state can't be modified at runtime. An immutable link is always enabled.
MEDIA_LNK_FL_DYNAMIC	The link enabled state can be modified during streaming. This flag is set by drivers and is read-only for applications.
MEDIA_LNK_FL_LINK_TYPE	This is a bitmask that defines the type of the link. Currently, two types of links are supported: MEDIA_LNK_FL_DATA_LINK if the link is between two pads MEDIA_LNK_FL_INTERFACE_LINK if the link is between an interface and an entity

1.5.4 Function Reference

media open()

Name

media-open - Open a media device

Synopsis

```
#include <fcntl.h>
```

int **open**(const char *_device_name_, int _flags_)

Arguments

device_name Device to be opened.

flags Open flags. Access mode must be either O_RDONLY or O_RDWR. Other flags have no effect.

Description

To open a media device applications call *open()* with the desired device name. The function has no side effects; the device configuration remain unchanged.

When the device is opened in read-only mode, attempts to modify its configuration will result in an error, and `errno` will be set to EBADF.

Return Value

open() returns the new file descriptor on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EACCES The requested access to the file is not allowed.

EMFILE The process already has the maximum number of files open.

ENFILE The system limit on the total number of open files has been reached.

ENOMEM Insufficient kernel memory was available.

ENXIO No device corresponding to this device special file exists.

media close()

Name

media-close - Close a media device

Synopsis

```
#include <unistd.h>
```

int **close**(int *fd*)

Arguments

fd File descriptor returned by *open()*.

Description

Closes the media device. Resources associated with the file descriptor are freed. The device configuration remain unchanged.

Return Value

close() returns 0 on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EBADF `fd` is not a valid open file descriptor.

media ioctl()

Name

media-ioctl - Control a media device

Synopsis

```
#include <sys/ioctl.h>
```

int **ioctl**(int *fd*, int *request*, void **argp*)

Arguments

fd File descriptor returned by *open()*.

request Media ioctl request code as defined in the media.h header file, for example MEDIA_IOC_SETUP_LINK.

argp Pointer to a request-specific structure.

Description

The *ioctl()* function manipulates media device parameters. The argument `fd` must be an open file descriptor.

The ioctl `request` code specifies the media function to be called. It has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument `argp` in bytes.

Macros and structures definitions specifying media ioctl requests and their parameters are located in the media.h header file. All media ioctl requests, their respective function and parameters are specified in *Function Reference* .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

Request-specific error codes are listed in the individual requests descriptions.

When an ioctl that takes an output or read/write parameter fails, the parameter remains unmodified.

ioctl MEDIA_IOC_DEVICE_INFO

Name

MEDIA_IOC_DEVICE_INFO - Query device information

Synopsis

int **ioctl**(int *fd*, *MEDIA_IOC_DEVICE_INFO*, struct *media_device_info* **argp*)

Arguments

fd File descriptor returned by *open()* .

argp

Description

All media devices must support the MEDIA_IOC_DEVICE_INFO ioctl. To query device information, applications call the ioctl with a pointer to a struct *media_device_info*. The driver fills the structure and returns the information to the application. The ioctl never fails.

media_device_info

Table 1.204: struct media_device_info

char	driver[16]	Name of the driver implementing the media API as a NUL-terminated ASCII string. The driver version is stored in the driver_version field. Driver specific applications can use this information to verify the driver identity. It is also useful to work around known bugs, or to identify drivers in error reports.
char	model[32]	Device model name as a NUL-terminated UTF-8 string. The device version is stored in the device_version field and is not be appended to the model name.
char	serial[40]	Serial number as a NUL-terminated ASCII string.
char	bus_info[32]	Location of the device in the system as a NUL-terminated ASCII string. This includes the bus type name (PCI, USB, ...) and a bus-specific identifier.
__u32	media_version	Media API version, formatted with the KERNEL_VERSION() macro.
__u32	hw_revision	Hardware device revision in a driver-specific format.
__u32	driver_version	Media device driver version, formatted with the KERNEL_VERSION() macro. Together with the driver field this identifies a particular driver.
__u32	reserved[31]	Reserved for future extensions. Drivers and applications must set this array to zero.

The serial and bus_info fields can be used to distinguish between multiple instances of otherwise identical hardware. The serial number takes precedence when provided and can be assumed to be unique. If the serial number is an empty string, the bus_info field can be used instead. The bus_info field is guaranteed to be unique, but can vary across reboots or device unplug/replug.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl MEDIA_IOC_G_TOPOLOGY

Name

MEDIA_IOC_G_TOPOLOGY - Enumerate the graph topology and graph element properties

Synopsis

int **ioctl**(int *fd*, *MEDIA_IOC_G_TOPOLOGY*, struct *media_v2_topology* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

The typical usage of this ioctl is to call it twice. On the first call, the structure defined at struct *media_v2_topology* should be zeroed. At return, if no errors happen, this ioctl will return the topology_version and the total number of entities, interfaces, pads and links.

Before the second call, the userspace should allocate arrays to store the graph elements that are desired, putting the pointers to them at the ptr_entities, ptr_interfaces, ptr_links and/or ptr_pads, keeping the other values untouched.

If the topology_version remains the same, the ioctl should fill the desired arrays with the media graph elements.

media_v2_topology

Table 1.205: struct media_v2_topology

__u64	topology_version	Version of the media graph topology. When the graph is created, this field starts with zero. Every time a graph element is added or removed, this field is incremented.
__u64	num_entities	Number of entities in the graph
__u64	ptr_entities	A pointer to a memory area where the entities array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the entities. It will just update num_entities
__u64	num_interfaces	Number of interfaces in the graph
__u64	ptr_interfaces	A pointer to a memory area where the interfaces array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the interfaces. It will just update num_interfaces
__u64	num_pads	Total number of pads in the graph
__u64	ptr_pads	A pointer to a memory area where the pads array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the pads. It will just update num_pads
__u64	num_links	Total number of data and interface links in the graph
__u64	ptr_links	A pointer to a memory area where the links array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the links. It will just update num_links

media_v2_entity

Table 1.206: struct media_v2_entity

__u32	id	Unique ID for the entity.
char	name[64]	Entity name as an UTF-8 NULL-terminated string.
__u32	function	Entity main function, see *Media entity types* for details.
__u32	reserved[12]	Reserved for future extensions. Drivers and applications must set this array to zero.

media_v2_interface

Table 1.207: struct media_v2_interface

__u32	id	Unique ID for the interface.
__u32	intf_type	Interface type, see *Media interface types* for details.
__u32	flags	Interface flags. Currently unused.
__u32	reserved[9]	Reserved for future extensions. Drivers and applications must set this array to zero.
struct media_v2_intf_devnode	devnode	Used only for device node interfaces. See *media_v2_intf_devnode* for details..

media_v2_intf_devnode

Table 1.208: struct media_v2_interface

__u32	major	Device node major number.
__u32	minor	Device node minor number.

media_v2_pad

Table 1.209: struct media_v2_pad

__u32	id	Unique ID for the pad.
__u32	entity_id	Unique ID for the entity where this pad belongs.
__u32	flags	Pad flags, see *Media pad flags* for more details.
__u32	reserved[9]	Reserved for future extensions. Drivers and applications must set this array to zero.

media_v2_link

Table 1.210: struct media_v2_pad

__u32	id	Unique ID for the pad.
__u32	source_id	On pad to pad links: unique ID for the source pad. On interface to entity links: unique ID for the interface.
__u32	sink_id	On pad to pad links: unique ID for the sink pad. On interface to entity links: unique ID for the entity.
__u32	flags	Link flags, see *Media link flags* for more details.
__u32	reserved[5]	Reserved for future extensions. Drivers and applications must set this array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENOSPC This is returned when either one or more of the num_entities, num_interfaces, num_links or num_pads are non-zero and are smaller than the actual number of elements inside the graph. This

may happen if the topology_version changed when compared to the last time this ioctl was called. Userspace should usually free the area for the pointers, zero the struct elements and call this ioctl again.

ioctl MEDIA_IOC_ENUM_ENTITIES

Name

MEDIA_IOC_ENUM_ENTITIES - Enumerate entities and their properties

Synopsis

int **ioctl**(int *fd*, *MEDIA_IOC_ENUM_ENTITIES*, struct *media_entity_desc* **argp*)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To query the attributes of an entity, applications set the id field of a struct *media_entity_desc* structure and call the MEDIA_IOC_ENUM_ENTITIES ioctl with a pointer to this structure. The driver fills the rest of the structure or returns an EINVAL error code when the id is invalid. Entities can be enumerated by or'ing the id with the MEDIA_ENT_ID_FLAG_NEXT flag. The driver will return information about the entity with the smallest id strictly larger than the requested one ('next entity'), or the EINVAL error code if there is none.

Entity IDs can be non-contiguous. Applications must *not* try to enumerate entities by calling MEDIA_IOC_ENUM_ENTITIES with increasing id's until they get an error.

media_entity_desc

Table 1.211: struct media_entity_desc

__u32	id			Entity id, set by the application. When the id is or'ed with MEDIA_ENT_ID_FLAG_NEXT, the driver clears the flag and returns the first entity with a larger id.
char	name[32]			Entity name as an UTF-8 NULL-terminated string.
__u32	type			Entity type, see *Media entity types* for details.
__u32	revision			Entity revision. Always zero (obsolete)
__u32	flags			Entity flags, see *Media entity flags* for details.
__u32	group_id			Entity group ID. Always zero (obsolete)
__u16	pads			Number of pads
__u16	links			Total number of outbound links. Inbound links are not counted in this field.
union				
	struct	dev		Valid for (sub-)devices that create a single device node.
		__u32	major	Device node major number.
		__u32	minor	Device node minor number.
	__u8	raw[184]		

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *media_entity_desc* `id` references a non-existing entity.

ioctl MEDIA_IOC_ENUM_LINKS

Name

MEDIA_IOC_ENUM_LINKS - Enumerate all pads and links for a given entity

Synopsis

int **ioctl**(int *fd*, *MEDIA_IOC_ENUM_LINKS*, struct *media_links_enum* *argp)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

To enumerate pads and/or links for a given entity, applications set the entity field of a struct *media_links_enum* structure and initialize the struct *media_pad_desc* and struct *media_link_desc* structure arrays pointed by the `pads` and `links` fields. They then call the MEDIA_IOC_ENUM_LINKS ioctl with a pointer to this structure.

If the `pads` field is not NULL, the driver fills the `pads` array with information about the entity's pads. The array must have enough room to store all the entity's pads. The number of pads can be retrieved with *ioctl MEDIA_IOC_ENUM_ENTITIES* .

If the `links` field is not NULL, the driver fills the `links` array with information about the entity's outbound links. The array must have enough room to store all the entity's outbound links. The number of outbound links can be retrieved with *ioctl MEDIA_IOC_ENUM_ENTITIES* .

Only forward links that originate at one of the entity's source pads are returned during the enumeration process.

media_links_enum

Table 1.212: struct media_links_enum

__u32	`entity`	Entity id, set by the application.
struct *media_pad_desc*	*`pads`	Pointer to a pads array allocated by the application. Ignored if NULL.
struct *media_link_desc*	*`links`	Pointer to a links array allocated by the application. Ignored if NULL.

media_pad_desc

Table 1.213: struct media_pad_desc

__u32	entity	ID of the entity this pad belongs to.
__u16	index	0-based pad index.
__u32	flags	Pad flags, see *Media pad flags* for more details.

media_link_desc

Table 1.214: struct media_link_desc

struct *media_pad_desc*	source	Pad at the origin of this link.
struct *media_pad_desc*	sink	Pad at the target of this link.
__u32	flags	Link flags, see *Media link flags* for more details.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *media_links_enum* id references a non-existing entity.

ioctl MEDIA_IOC_SETUP_LINK

Name

MEDIA_IOC_SETUP_LINK - Modify the properties of a link

Synopsis

int **ioctl**(int *fd*, *MEDIA_IOC_SETUP_LINK*, struct *media_link_desc* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

To change link properties applications fill a struct *media_link_desc* with link identification information (source and sink pad) and the new requested link flags. They then call the MEDIA_IOC_SETUP_LINK ioctl with a pointer to that structure.

The only configurable property is the ENABLED link flag to enable/disable a link. Links marked with the IMMUTABLE link flag can not be enabled or disabled.

Link configuration has no side effect on other links. If an enabled link at the sink pad prevents the link from being enabled, the driver returns with an EBUSY error code.

Only links marked with the DYNAMIC link flag can be enabled/disabled while streaming media data. Attempting to enable or disable a streaming non-dynamic link will return an EBUSY error code.

If the specified link can't be found the driver returns with an EINVAL error code.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct *media_link_desc* references a non-existing link, or the link is immutable and an attempt to modify its configuration was made.

1.5.5 Media Controller Header File

media.h

```
/*
 * Multimedia device API
 *
 * Copyright (C) 2010 Nokia Corporation
 *
 * Contacts: Laurent Pinchart <laurent.pinchart@ideasonboard.com>
 *           Sakari Ailus <sakari.ailus@iki.fi>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#ifndef __LINUX_MEDIA_H
#define __LINUX_MEDIA_H

#ifndef __KERNEL__
#include <stdint.h>
#endif
#include <linux/ioctl.h>
#include <linux/types.h>
#include <linux/version.h>

#define MEDIA_API_VERSION       KERNEL_VERSION(0, 1, 0)

struct  media_device_info
 {
        char driver[16];
        char model[32];
        char serial[40];
        char bus_info[32];
        __u32 media_version;
        __u32 hw_revision;
        __u32 driver_version;
        __u32 reserved[31];
};
```

```
#define  MEDIA_ENT_ID_FLAG_NEXT           (1 << 31)

/*
 * Initial value to be used when a new entity is created
 * Drivers should change it to something useful
 */
#define  MEDIA_ENT_F_UNKNOWN      0x00000000

/*
 * Base number ranges for entity functions
 *
 * NOTE: those ranges and entity function number are phased just to
 * make it easier to maintain this file. Userspace should not rely on
 * the ranges to identify a group of function types, as newer
 * functions can be added with any name within the full u32 range.
 */
#define MEDIA_ENT_F_BASE                0x00000000
#define MEDIA_ENT_F_OLD_BASE            0x00010000
#define MEDIA_ENT_F_OLD_SUBDEV_BASE     0x00020000

/*
 * DVB entities
 */
#define  MEDIA_ENT_F_DTV_DEMOD            (MEDIA_ENT_F_BASE + 0x00001)
#define  MEDIA_ENT_F_TS_DEMUX             (MEDIA_ENT_F_BASE + 0x00002)
#define  MEDIA_ENT_F_DTV_CA               (MEDIA_ENT_F_BASE + 0x00003)
#define  MEDIA_ENT_F_DTV_NET_DECAP        (MEDIA_ENT_F_BASE + 0x00004)

/*
 * I/O entities
 */
#define  MEDIA_ENT_F_IO_DTV               (MEDIA_ENT_F_BASE + 0x01001)
#define  MEDIA_ENT_F_IO_VBI               (MEDIA_ENT_F_BASE + 0x01002)
#define  MEDIA_ENT_F_IO_SWRADIO           (MEDIA_ENT_F_BASE + 0x01003)

/*
 * Analog TV IF-PLL decoders
 *
 * It is a responsibility of the master/bridge drivers to create links
 * for  MEDIA_ENT_F_IF_VID_DECODER  and MEDIA_ENT_F_IF_AUD_DECODER.
 */
#define  MEDIA_ENT_F_IF_VID_DECODER       (MEDIA_ENT_F_BASE + 0x02001)
#define  MEDIA_ENT_F_IF_AUD_DECODER       (MEDIA_ENT_F_BASE + 0x02002)

/*
 * Audio Entity Functions
 */
#define  MEDIA_ENT_F_AUDIO_CAPTURE        (MEDIA_ENT_F_BASE + 0x03001)
#define  MEDIA_ENT_F_AUDIO_PLAYBACK       (MEDIA_ENT_F_BASE + 0x03002)
#define  MEDIA_ENT_F_AUDIO_MIXER          (MEDIA_ENT_F_BASE + 0x03003)

/*
 * Processing entities
 */
#define  MEDIA_ENT_F_PROC_VIDEO_COMPOSER          (MEDIA_ENT_F_BASE + 0x4001)
#define  MEDIA_ENT_F_PROC_VIDEO_PIXEL_FORMATTER   (MEDIA_ENT_F_BASE + 0x4002)
#define  MEDIA_ENT_F_PROC_VIDEO_PIXEL_ENC_CONV    (MEDIA_ENT_F_BASE + 0x4003)
```

```
#define  MEDIA_ENT_F_PROC_VIDEO_LUT              (MEDIA_ENT_F_BASE + 0x4004)
#define  MEDIA_ENT_F_PROC_VIDEO_SCALER           (MEDIA_ENT_F_BASE + 0x4005)
#define  MEDIA_ENT_F_PROC_VIDEO_STATISTICS       (MEDIA_ENT_F_BASE + 0x4006)

/*
 * Connectors
 */
/* It is a responsibility of the entity drivers to add connectors and links */
#ifdef __KERNEL__
        /*
         * For now, it should not be used in userspace, as some
         * definitions may change
         */

#define  MEDIA_ENT_F_CONN_RF             (MEDIA_ENT_F_BASE + 0x30001)
#define  MEDIA_ENT_F_CONN_SVIDEO         (MEDIA_ENT_F_BASE + 0x30002)
#define  MEDIA_ENT_F_CONN_COMPOSITE      (MEDIA_ENT_F_BASE + 0x30003)

#endif

/*
 * Don't touch on those. The ranges MEDIA_ENT_F_OLD_BASE and
 * MEDIA_ENT_F_OLD_SUBDEV_BASE are kept to keep backward compatibility
 * with the legacy v1 API.The number range is out of range by purpose:
 * several previously reserved numbers got excluded from this range.
 *
 * Subdevs are initialized with MEDIA_ENT_T_V4L2_SUBDEV_UNKNOWN,
 * in order to preserve backward compatibility.
 * Drivers must change to the proper subdev type before
 * registering the entity.
 */

#define  MEDIA_ENT_F_IO_V4L              (MEDIA_ENT_F_OLD_BASE + 1)

#define  MEDIA_ENT_F_CAM_SENSOR          (MEDIA_ENT_F_OLD_SUBDEV_BASE + 1)
#define  MEDIA_ENT_F_FLASH               (MEDIA_ENT_F_OLD_SUBDEV_BASE + 2)
#define  MEDIA_ENT_F_LENS                (MEDIA_ENT_F_OLD_SUBDEV_BASE + 3)
#define  MEDIA_ENT_F_ATV_DECODER         (MEDIA_ENT_F_OLD_SUBDEV_BASE + 4)
/*
 * It is a responsibility of the master/bridge drivers to add connectors
 * and links for MEDIA_ENT_F_TUNER. Please notice that some old tuners
 * may require the usage of separate I2C chips to decode analog TV signals,
 * when the master/bridge chipset doesn't have its own TV standard decoder.
 * On such cases, the IF-PLL staging is mapped via one or two entities:
 *  MEDIA_ENT_F_IF_VID_DECODER  and/or MEDIA_ENT_F_IF_AUD_DECODER.
 */
#define  MEDIA_ENT_F_TUNER               (MEDIA_ENT_F_OLD_SUBDEV_BASE + 5)

#define  MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN  MEDIA_ENT_F_OLD_SUBDEV_BASE

#if !defined(__KERNEL__) || defined(__NEED_MEDIA_LEGACY_API)

/*
 * Legacy symbols used to avoid userspace compilation breakages
 *
 * Those symbols map the entity function into types and should be
 * used only on legacy programs for legacy hardware. Don't rely
 * on those for MEDIA_IOC_G_TOPOLOGY.
```

```
 */
#define MEDIA_ENT_TYPE_SHIFT            16
#define MEDIA_ENT_TYPE_MASK             0x00ff0000
#define MEDIA_ENT_SUBTYPE_MASK          0x0000ffff

/* End of the old subdev reserved numberspace */
#define MEDIA_ENT_T_DEVNODE_UNKNOWN     (MEDIA_ENT_T_DEVNODE | \
                                         MEDIA_ENT_SUBTYPE_MASK)

#define MEDIA_ENT_T_DEVNODE             MEDIA_ENT_F_OLD_BASE
#define MEDIA_ENT_T_DEVNODE_V4L          MEDIA_ENT_F_IO_V4L
#define MEDIA_ENT_T_DEVNODE_FB          (MEDIA_ENT_T_DEVNODE + 2)
#define MEDIA_ENT_T_DEVNODE_ALSA        (MEDIA_ENT_T_DEVNODE + 3)
#define MEDIA_ENT_T_DEVNODE_DVB         (MEDIA_ENT_T_DEVNODE + 4)

#define MEDIA_ENT_T_UNKNOWN              MEDIA_ENT_F_UNKNOWN
#define MEDIA_ENT_T_V4L2_VIDEO           MEDIA_ENT_F_IO_V4L
#define MEDIA_ENT_T_V4L2_SUBDEV          MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN
#define MEDIA_ENT_T_V4L2_SUBDEV_SENSOR   MEDIA_ENT_F_CAM_SENSOR
#define MEDIA_ENT_T_V4L2_SUBDEV_FLASH    MEDIA_ENT_F_FLASH
#define MEDIA_ENT_T_V4L2_SUBDEV_LENS     MEDIA_ENT_F_LENS
#define MEDIA_ENT_T_V4L2_SUBDEV_DECODER  MEDIA_ENT_F_ATV_DECODER
#define MEDIA_ENT_T_V4L2_SUBDEV_TUNER    MEDIA_ENT_F_TUNER
#endif

/* Entity flags */
#define  MEDIA_ENT_FL_DEFAULT             (1 << 0)
#define  MEDIA_ENT_FL_CONNECTOR           (1 << 1)

struct  media_entity_desc
 {
        __u32 id;
        char name[32];
        __u32 type;
        __u32 revision;
        __u32 flags;
        __u32 group_id;
        __u16 pads;
        __u16 links;

        __u32 reserved[4];

        union {
                /* Node specifications */
                struct {
                        __u32 major;
                        __u32 minor;
                } dev;

#if 1
                /*
                 * TODO: this shouldn't have been added without
                 * actual drivers that use this. When the first real driver
                 * appears that sets this information, special attention
                 * should be given whether this information is 1) enough, and
                 * 2) can deal with udev rules that rename devices. The struct
                 * dev would not be sufficient for this since that does not
                 * contain the subdevice information. In addition, struct dev
```

```
                 * can only refer to a single device, and not to multiple (e.g.
                 * pcm and mixer devices).
                 *
                 * So for now mark this as a to do.
                 */
                struct {
                        __u32 card;
                        __u32 device;
                        __u32 subdevice;
                } alsa;
#endif

#if 1
                /*
                 * DEPRECATED: previous node specifications. Kept just to
                 * avoid breaking compilation, but media_entity_desc.dev
                 * should be used instead. In particular, alsa and dvb
                 * fields below are wrong: for all devnodes, there should
                 * be just major/minor inside the struct, as this is enough
                 * to represent any devnode, no matter what type.
                 */
                struct {
                        __u32 major;
                        __u32 minor;
                } v4l;
                struct {
                        __u32 major;
                        __u32 minor;
                } fb;
                int dvb;
#endif

                /* Sub-device specifications */
                /* Nothing needed yet */
                __u8 raw[184];
        };
};

#define  MEDIA_PAD_FL_SINK                (1 << 0)
#define  MEDIA_PAD_FL_SOURCE              (1 << 1)
#define  MEDIA_PAD_FL_MUST_CONNECT        (1 << 2)

struct  media_pad_desc
 {
        __u32 entity;           /* entity ID */
        __u16 index;            /* pad index */
        __u32 flags;            /* pad flags */
        __u32 reserved[2];
};

#define  MEDIA_LNK_FL_ENABLED             (1 << 0)
#define  MEDIA_LNK_FL_IMMUTABLE           (1 << 1)
#define  MEDIA_LNK_FL_DYNAMIC             (1 << 2)

#define  MEDIA_LNK_FL_LINK_TYPE           (0xf << 28)
#  define  MEDIA_LNK_FL_DATA_LINK         (0 << 28)
#  define  MEDIA_LNK_FL_INTERFACE_LINK    (1 << 28)
```

```
struct  media_link_desc
 {
        struct  media_pad_desc
 source;
        struct  media_pad_desc
 sink;
        __u32 flags;
        __u32 reserved[2];
};

struct  media_links_enum
 {
        __u32 entity;
        /* Should have enough room for pads elements */
        struct  media_pad_desc
 __user *pads;
        /* Should have enough room for links elements */
        struct  media_link_desc
 __user *links;
        __u32 reserved[4];
};

/* Interface type ranges */

#define MEDIA_INTF_T_DVB_BASE   0x00000100
#define MEDIA_INTF_T_V4L_BASE   0x00000200
#define MEDIA_INTF_T_ALSA_BASE  0x00000300

/* Interface types */

#define  MEDIA_INTF_T_DVB_FE      (MEDIA_INTF_T_DVB_BASE)
#define  MEDIA_INTF_T_DVB_DEMUX   (MEDIA_INTF_T_DVB_BASE + 1)
#define  MEDIA_INTF_T_DVB_DVR     (MEDIA_INTF_T_DVB_BASE + 2)
#define  MEDIA_INTF_T_DVB_CA      (MEDIA_INTF_T_DVB_BASE + 3)
#define  MEDIA_INTF_T_DVB_NET     (MEDIA_INTF_T_DVB_BASE + 4)

#define  MEDIA_INTF_T_V4L_VIDEO   (MEDIA_INTF_T_V4L_BASE)
#define  MEDIA_INTF_T_V4L_VBI     (MEDIA_INTF_T_V4L_BASE + 1)
#define  MEDIA_INTF_T_V4L_RADIO   (MEDIA_INTF_T_V4L_BASE + 2)
#define  MEDIA_INTF_T_V4L_SUBDEV  (MEDIA_INTF_T_V4L_BASE + 3)
#define  MEDIA_INTF_T_V4L_SWRADIO  (MEDIA_INTF_T_V4L_BASE + 4)
#define  MEDIA_INTF_T_V4L_TOUCH   (MEDIA_INTF_T_V4L_BASE + 5)

#define  MEDIA_INTF_T_ALSA_PCM_CAPTURE    (MEDIA_INTF_T_ALSA_BASE)
#define  MEDIA_INTF_T_ALSA_PCM_PLAYBACK   (MEDIA_INTF_T_ALSA_BASE + 1)
#define  MEDIA_INTF_T_ALSA_CONTROL        (MEDIA_INTF_T_ALSA_BASE + 2)
#define  MEDIA_INTF_T_ALSA_COMPRESS       (MEDIA_INTF_T_ALSA_BASE + 3)
#define  MEDIA_INTF_T_ALSA_RAWMIDI        (MEDIA_INTF_T_ALSA_BASE + 4)
#define  MEDIA_INTF_T_ALSA_HWDEP          (MEDIA_INTF_T_ALSA_BASE + 5)
#define  MEDIA_INTF_T_ALSA_SEQUENCER      (MEDIA_INTF_T_ALSA_BASE + 6)
#define  MEDIA_INTF_T_ALSA_TIMER          (MEDIA_INTF_T_ALSA_BASE + 7)

/*
 * MC next gen API definitions
 *
 * NOTE: The declarations below are close to the MC RFC for the Media
 *       Controller, the next generation. Yet, there are a few adjustments
 *       to do, as we want to be able to have a functional API before
```

```
 *       the MC properties change. Those will be properly marked below.
 *       Please also notice that I removed ``num_pads'', ``num_links'',
 *       from the proposal, as a proper userspace application will likely
 *       use lists for pads/links, just as we intend to do in Kernelspace.
 *       The API definition should be freed from fields that are bound to
 *       some specific data structure.
 *
 * FIXME: Currently, I opted to name the new types as ``media_v2'', as this
 *        won't cause any conflict with the Kernelspace namespace, nor with
 *        the previous kAPI media_*_desc namespace. This can be changed
 *        later, before the adding this API upstream.
 */

struct  media_v2_entity
 {
        __u32 id;
        char name[64];          /* FIXME: move to a property? (RFC says so) */
        __u32 function;         /* Main function of the entity */
        __u32 reserved[6];
} __attribute__ ((packed));

/* Should match the specific fields at media_intf_devnode */
struct  media_v2_intf_devnode
 {
        __u32 major;
        __u32 minor;
} __attribute__ ((packed));

struct  media_v2_interface
 {
        __u32 id;
        __u32 intf_type;
        __u32 flags;
        __u32 reserved[9];

        union {
                struct  media_v2_intf_devnode
 devnode;
                __u32 raw[16];
        };
} __attribute__ ((packed));

struct  media_v2_pad
 {
        __u32 id;
        __u32 entity_id;
        __u32 flags;
        __u32 reserved[5];
} __attribute__ ((packed));

struct  media_v2_link
 {
        __u32 id;
        __u32 source_id;
        __u32 sink_id;
        __u32 flags;
        __u32 reserved[6];
} __attribute__ ((packed));
```

```
struct  media_v2_topology
 {
        __u64 topology_version;

        __u32 num_entities;
        __u32 reserved1;
        __u64 ptr_entities;

        __u32 num_interfaces;
        __u32 reserved2;
        __u64 ptr_interfaces;

        __u32 num_pads;
        __u32 reserved3;
        __u64 ptr_pads;

        __u32 num_links;
        __u32 reserved4;
        __u64 ptr_links;
} __attribute__ ((packed));

/* ioctls */

#define  MEDIA_IOC_DEVICE_INFO            _IOWR(`|', 0x00, struct  media_device_info
)
#define  MEDIA_IOC_ENUM_ENTITIES          _IOWR(`|', 0x01, struct  media_entity_desc
)
#define  MEDIA_IOC_ENUM_LINKS             _IOWR(`|', 0x02, struct  media_links_enum
)
#define  MEDIA_IOC_SETUP_LINK             _IOWR(`|', 0x03, struct  media_link_desc
)
#define  MEDIA_IOC_G_TOPOLOGY             _IOWR(`|', 0x04, struct  media_v2_topology
)

#endif /* __LINUX_MEDIA_H */
```

1.5.6 Revision and Copyright

Authors:

- Pinchart, Laurent <laurent.pinchart@ideasonboard.com>
- Initial version.
- Carvalho Chehab, Mauro <mchehab@kernel.org>
- MEDIA_IOC_G_TOPOLOGY documentation and documentation improvements.

1.5.7 Revision History

revision 1.1.0 / 2015-12-12 (*mcc*)

revision 1.0.0 / 2010-11-10 (*lp*)

Initial revision

1.6 Part V - Consumer Electronics Control API

This part describes the CEC: Consumer Electronics Control

Table of Contents

1.6.1 Introduction

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

HDMI connectors provide a single pin for use by the Consumer Electronics Control protocol. This protocol allows different devices connected by an HDMI cable to communicate. The protocol for CEC version 1.4 is defined in supplements 1 (CEC) and 2 (HEAC or HDMI Ethernet and Audio Return Channel) of the HDMI 1.4a (*HDMI*) specification and the extensions added to CEC version 2.0 are defined in chapter 11 of the HDMI 2.0 (*HDMI2*) specification.

The bitrate is very slow (effectively no more than 36 bytes per second) and is based on the ancient AV.link protocol used in old SCART connectors. The protocol closely resembles a crazy Rube Goldberg contraption and is an unholy mix of low and high level messages. Some messages, especially those part of the HEAC protocol layered on top of CEC, need to be handled by the kernel, others can be handled either by the kernel or by userspace.

In addition, CEC can be implemented in HDMI receivers, transmitters and in USB devices that have an HDMI input and an HDMI output and that control just the CEC pin.

Drivers that support CEC will create a CEC device node (/dev/cecX) to give userspace access to the CEC adapter. The *ioctl CEC_ADAP_G_CAPS* ioctl will tell userspace what it is allowed to do.

1.6.2 Function Reference

cec open()

Name

cec-open - Open a cec device

Synopsis

```
#include <fcntl.h>
```

int **open**(const char ***device_name*, int *flags*)

Arguments

`device_name` Device to be opened.

`flags` Open flags. Access mode must be `O_RDWR`.

When the `O_NONBLOCK` flag is given, the *CEC_RECEIVE* and *CEC_DQEVENT()* ioctls will return the `EAGAIN` error code when no message or event is available, and ioctls *CEC_TRANSMIT* , *CEC_ADAP_S_PHYS_ADDR* and *CEC_ADAP_S_LOG_ADDRS* all return 0.

Other flags have no effect.

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

To open a cec device applications call *open()* with the desired device name. The function has no side effects; the device configuration remain unchanged.

When the device is opened in read-only mode, attempts to modify its configuration will result in an error, and `errno` will be set to EBADF.

Return Value

open() returns the new file descriptor on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes include:

EACCES The requested access to the file is not allowed.

EMFILE The process already has the maximum number of files open.

ENFILE The system limit on the total number of open files has been reached.

ENOMEM Insufficient kernel memory was available.

ENXIO No device corresponding to this device special file exists.

cec close()

Name

cec-close - Close a cec device

Synopsis

```
#include <unistd.h>
```

int **close**(int *fd*)

Arguments

fd File descriptor returned by *open()*.

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

Closes the cec device. Resources associated with the file descriptor are freed. The device configuration remain unchanged.

Return Value

close() returns 0 on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EBADF `fd` is not a valid open file descriptor.

cec ioctl()

Name

cec-ioctl - Control a cec device

Synopsis

```
#include <sys/ioctl.h>
```

int **ioctl**(int *fd*, int *request*, void **argp*)

Arguments

fd File descriptor returned by *open()*.

request CEC ioctl request code as defined in the cec.h header file, for example *CEC_ADAP_G_CAPS()*.

argp Pointer to a request-specific structure.

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

The `ioctl()` function manipulates cec device parameters. The argument `fd` must be an open file descriptor.

The ioctl `request` code specifies the cec function to be called. It has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument `argp` in bytes.

Macros and structures definitions specifying cec ioctl requests and their parameters are located in the cec.h header file. All cec ioctl requests, their respective function and parameters are specified in *Function Reference* .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

Request-specific error codes are listed in the individual requests descriptions.

When an ioctl that takes an output or read/write parameter fails, the parameter remains unmodified.

cec poll()

Name

cec-poll - Wait for some event on a file descriptor

Synopsis

```
#include <sys/poll.h>
```

int **poll**(struct pollfd **ufds*, unsigned int *nfds*, int *timeout*)

Arguments

`ufds` List of FD events to be watched

`nfds` Number of FD efents at the *ufds array

`timeout` Timeout to wait for events

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

With the `poll()` function applications can wait for CEC events.

On success `poll()` returns the number of file descriptors that have been selected (that is, file descriptors for which the `revents` field of the respective struct `pollfd` is non-zero). CEC devices set the `POLLIN` and `POLLRDNORM` flags in the `revents` field if there are messages in the receive queue. If the transmit queue has room for new messages, the `POLLOUT` and `POLLWRNORM` flags are set. If there are events in the event queue, then the `POLLPRI` flag is set. When the function timed out it returns a value of zero, on failure it returns -1 and the `errno` variable is set appropriately.

For more details see the `poll()` manual page.

Return Value

On success, `poll()` returns the number structures which have non-zero `revents` fields, or zero if the call timed out. On error -1 is returned, and the `errno` variable is set appropriately:

EBADF One or more of the `ufds` members specify an invalid file descriptor.

EFAULT `ufds` references an inaccessible memory area.

EINTR The call was interrupted by a signal.

EINVAL The `nfds` argument is greater than `OPEN_MAX`.

ioctl CEC_ADAP_G_CAPS

Name

CEC_ADAP_G_CAPS - Query device capabilities

Synopsis

int **ioctl**(int *fd*, *CEC_ADAP_G_CAPS*, struct *cec_caps* **argp*)

Arguments

fd File descriptor returned by *open()* .

`argp`

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

All cec devices must support *ioctl CEC_ADAP_G_CAPS* . To query device information, applications call the ioctl with a pointer to a struct *cec_caps*. The driver fills the structure and returns the information to the application. The ioctl never fails.

cec_caps

Table 1.215: struct cec_caps

char	`driver[32]`	The name of the cec adapter driver.
char	`name[32]`	The name of this CEC adapter. The combination `driver` and `name` must be unique.
__u32	`capabilities`	The capabilities of the CEC adapter, see *CEC Capabilities Flags* .
__u32	`version`	CEC Framework API version, formatted with the `KERNEL_VERSION()` macro.

Table 1.216: CEC Capabilities Flags

CEC_CAP_PHYS_ADDR	0x00000001	Userspace has to configure the physical address by calling *ioctl CEC_ADAP_S_PHYS_ADDR* . If this capability isn't set, then setting the physical address is handled by the kernel whenever the EDID is set (for an HDMI receiver) or read (for an HDMI transmitter).
CEC_CAP_LOG_ADDRS	0x00000002	Userspace has to configure the logical addresses by calling *ioctl CEC_ADAP_S_LOG_ADDRS* . If this capability isn't set, then the kernel will have configured this.
CEC_CAP_TRANSMIT	0x00000004	Userspace can transmit CEC messages by calling *ioctl CEC_TRANSMIT* . This implies that userspace can be a follower as well, since being able to transmit messages is a prerequisite of becoming a follower. If this capability isn't set, then the kernel will handle all CEC transmits and process all CEC messages it receives.
CEC_CAP_PASSTHROUGH	0x00000008	Userspace can use the passthrough mode by calling *ioctl CEC_S_MODE* .
CEC_CAP_RC	0x00000010	This adapter supports the remote control protocol.
CEC_CAP_MONITOR_ALL	0x00000020	The CEC hardware can monitor all messages, not just directed and broadcast messages.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls CEC_ADAP_G_LOG_ADDRS and CEC_ADAP_S_LOG_ADDRS

Name

CEC_ADAP_G_LOG_ADDRS, CEC_ADAP_S_LOG_ADDRS - Get or set the logical addresses

Synopsis

int **ioctl**(int *fd*, *CEC_ADAP_G_LOG_ADDRS*, struct *cec_log_addrs* *argp)

int **ioctl**(int *fd*, *CEC_ADAP_S_LOG_ADDRS*, struct *cec_log_addrs* *argp)

Arguments

fd File descriptor returned by *open()*.

argp Pointer to struct *cec_log_addrs*.

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

To query the current CEC logical addresses, applications call *ioctl CEC_ADAP_G_LOG_ADDRS* with a pointer to a struct *cec_log_addrs* where the driver stores the logical addresses.

To set new logical addresses, applications fill in struct *cec_log_addrs* and call *ioctl CEC_ADAP_S_LOG_ADDRS* with a pointer to this struct. The *ioctl CEC_ADAP_S_LOG_ADDRS* is only available if CEC_CAP_LOG_ADDRS is set (the ENOTTY error code is returned otherwise). The *ioctl CEC_ADAP_S_LOG_ADDRS* can only be called by a file descriptor in initiator mode (see *ioctls CEC_G_MODE and CEC_S_MODE*), if not the EBUSY error code will be returned.

To clear existing logical addresses set num_log_addrs to 0. All other fields will be ignored in that case. The adapter will go to the unconfigured state.

If the physical address is valid (see *ioctl CEC_ADAP_S_PHYS_ADDR*), then this ioctl will block until all requested logical addresses have been claimed. If the file descriptor is in non-blocking mode then it will not wait for the logical addresses to be claimed, instead it just returns 0.

A *CEC_EVENT_STATE_CHANGE* event is sent when the logical addresses are claimed or cleared.

Attempting to call *ioctl CEC_ADAP_S_LOG_ADDRS* when logical address types are already defined will return with error EBUSY.

cec_log_addrs

Table 1.217: struct cec_log_addrs

__u8	log_addr[CEC_MAX_LOG_ADDRS]	The actual logical addresses that were claimed. This is set by the driver. If no logical address could be claimed, then it is set to CEC_LOG_ADDR_INVALID. If this adapter is Unregistered, then log_addr[0] is set to 0xf and all others to CEC_LOG_ADDR_INVALID.
__u16	log_addr_mask	The bitmask of all logical addresses this adapter has claimed. If this adapter is Unregistered then log_addr_mask sets bit 15 and clears all other bits. If this adapter is not configured at all, then log_addr_mask is set to 0. Set by the driver.
__u8	cec_version	The CEC version that this adapter shall use. See *CEC Versions* . Used to implement the CEC_MSG_CEC_VERSION and CEC_MSG_REPORT_FEATURES messages. Note that *CEC_OP_CEC_VERSION_1_3A* is not allowed by the CEC framework.

Continued on next page

Table 1.217 – continued from previous page

__u8	num_log_addrs	Number of logical addresses to set up. Must be ≤ available_log_addrs as returned by *ioctl CEC_ADAP_G_CAPS* . All arrays in this structure are only filled up to index available_log_addrs-1. The remaining array elements will be ignored. Note that the CEC 2.0 standard allows for a maximum of 2 logical addresses, although some hardware has support for more. CEC_MAX_LOG_ADDRS is 4. The driver will return the actual number of logical addresses it could claim, which may be less than what was requested. If this field is set to 0, then the CEC adapter shall clear all claimed logical addresses and all other fields will be ignored.
__u32	vendor_id	The vendor ID is a 24-bit number that identifies the specific vendor or entity. Based on this ID vendor specific commands may be defined. If you do not want a vendor ID then set it to CEC_VENDOR_ID_NONE.
__u32	flags	Flags. See *Flags for struct cec_log_addrs* for a list of available flags.
char	osd_name[15]	The On-Screen Display name as is returned by the CEC_MSG_SET_OSD_NAME message.
__u8	primary_device_type[CEC_MAX_LOG_ADDRS]	Primary device type for each logical address. See *CEC Primary Device Types* for possible types.
__u8	log_addr_type[CEC_MAX_LOG_ADDRS]	Logical address types. See *CEC Logical Address Types* for possible types. The driver will update this with the actual logical address type that it claimed (e.g. it may have to fallback to *CEC_LOG_ADDR_TYPE_UNREGISTERED*).
__u8	all_device_types[CEC_MAX_LOG_ADDRS]	CEC 2.0 specific: the bit mask of all device types. See *CEC All Device Types Flags* . It is used in the CEC 2.0 CEC_MSG_REPORT_FEATURES message. For CEC 1.4 you can either leave this field to 0, or fill it in according to the CEC 2.0 guidelines to give the CEC framework more information about the device type, even though the framework won't use it directly in the CEC message.
__u8	features[CEC_MAX_LOG_ADDRS][12]	Features for each logical address. It is used in the CEC 2.0 CEC_MSG_REPORT_FEATURES message. The 12 bytes include both the RC Profile and the Device Features. For CEC 1.4 you can either leave this field to all 0, or fill it in according to the CEC 2.0 guidelines to give the CEC framework more information about the device type, even though the framework won't use it directly in the CEC message.

Table 1.218: Flags for struct cec_log_addrs

CEC_LOG_ADDRS_FL_ALLOW_UNREG_FALLBACK	1	By default if no logical address of the requested type can be claimed, then it will go back to the unconfigured state. If this flag is set, then it will fallback to the Unregistered logical address. Note that if the Unregistered logical address was explicitly requested, then this flag has no effect.
CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU	2	By default the CEC_MSG_USER_CONTROL_PRESSED and CEC_MSG_USER_CONTROL_RELEASED messages are only passed on to the follower(s), if any. If this flag is set, then these messages are also passed on to the remote control input subsystem and will appear as keystrokes. This features needs to be enabled explicitly. If CEC is used to enter e.g. passwords, then you may not want to enable this to avoid trivial snooping of the keystrokes.
CEC_LOG_ADDRS_FL_CDC_ONLY	4	If this flag is set, then the device is CDC-Only. CDC-Only CEC devices are CEC devices that can only handle CDC messages. All other messages are ignored.

Table 1.219: CEC Versions

CEC_OP_CEC_VERSION_1_3A	4	CEC version according to the HDMI 1.3a standard.
CEC_OP_CEC_VERSION_1_4B	5	CEC version according to the HDMI 1.4b standard.
CEC_OP_CEC_VERSION_2_0	6	CEC version according to the HDMI 2.0 standard.

Table 1.220: CEC Primary Device Types

CEC_OP_PRIM_DEVTYPE_TV	0	Use for a TV.
CEC_OP_PRIM_DEVTYPE_RECORD	1	Use for a recording device.
CEC_OP_PRIM_DEVTYPE_TUNER	3	Use for a device with a tuner.
CEC_OP_PRIM_DEVTYPE_PLAYBACK	4	Use for a playback device.
CEC_OP_PRIM_DEVTYPE_AUDIOSYSTEM	5	Use for an audio system (e.g. an audio/video receiver).
CEC_OP_PRIM_DEVTYPE_SWITCH	6	Use for a CEC switch.
CEC_OP_PRIM_DEVTYPE_VIDEOPROC	7	Use for a video processor device.

Table 1.221: CEC Logical Address Types

CEC_LOG_ADDR_TYPE_TV	0	Use for a TV.
CEC_LOG_ADDR_TYPE_RECORD	1	Use for a recording device.
CEC_LOG_ADDR_TYPE_TUNER	2	Use for a tuner device.
CEC_LOG_ADDR_TYPE_PLAYBACK	3	Use for a playback device.
CEC_LOG_ADDR_TYPE_AUDIOSYSTEM	4	Use for an audio system device.
CEC_LOG_ADDR_TYPE_SPECIFIC	5	Use for a second TV or for a video processor device.
CEC_LOG_ADDR_TYPE_UNREGISTERED	6	Use this if you just want to remain unregistered. Used for pure CEC switches or CDC-only devices (CDC: Capability Discovery and Control).

Table 1.222: CEC All Device Types Flags

CEC_OP_ALL_DEVTYPE_TV	0x80	This supports the TV type.
CEC_OP_ALL_DEVTYPE_RECORD	0x40	This supports the Recording type.
CEC_OP_ALL_DEVTYPE_TUNER	0x20	This supports the Tuner type.
CEC_OP_ALL_DEVTYPE_PLAYBACK	0x10	This supports the Playback type.
CEC_OP_ALL_DEVTYPE_AUDIOSYSTEM	0x08	This supports the Audio System type.
CEC_OP_ALL_DEVTYPE_SWITCH	0x04	This supports the CEC Switch or Video Processing type.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls CEC_ADAP_G_PHYS_ADDR and CEC_ADAP_S_PHYS_ADDR

Name

CEC_ADAP_G_PHYS_ADDR, CEC_ADAP_S_PHYS_ADDR - Get or set the physical address

Synopsis

int **ioctl**(int *fd*, *CEC_ADAP_G_PHYS_ADDR*, __u16 *argp)

int **ioctl**(int *fd*, *CEC_ADAP_S_PHYS_ADDR*, __u16 *argp)

Arguments

fd File descriptor returned by *open()*.

argp Pointer to the CEC address.

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

To query the current physical address applications call *ioctl CEC_ADAP_G_PHYS_ADDR* with a pointer to a __u16 where the driver stores the physical address.

To set a new physical address applications store the physical address in a __u16 and call *ioctl CEC_ADAP_S_PHYS_ADDR* with a pointer to this integer. The *ioctl CEC_ADAP_S_PHYS_ADDR* is only available if CEC_CAP_PHYS_ADDR is set (the ENOTTY error code will be returned otherwise). The *ioctl CEC_ADAP_S_PHYS_ADDR* can only be called by a file descriptor in initiator mode (see *ioctls CEC_G_MODE and CEC_S_MODE*), if not the EBUSY error code will be returned.

To clear an existing physical address use CEC_PHYS_ADDR_INVALID. The adapter will go to the unconfigured state.

If logical address types have been defined (see *ioctl CEC_ADAP_S_LOG_ADDRS*), then this ioctl will block until all requested logical addresses have been claimed. If the file descriptor is in non-blocking mode then it will not wait for the logical addresses to be claimed, instead it just returns 0.

A *CEC_EVENT_STATE_CHANGE* event is sent when the physical address changes.

The physical address is a 16-bit number where each group of 4 bits represent a digit of the physical address a.b.c.d where the most significant 4 bits represent 'a'. The CEC root device (usually the TV) has address 0.0.0.0. Every device that is hooked up to an input of the TV has address a.0.0.0 (where 'a' is ≥ 1), devices hooked up to those in turn have addresses a.b.0.0, etc. So a topology of up to 5 devices deep is supported. The physical address a device shall use is stored in the EDID of the sink.

For example, the EDID for each HDMI input of the TV will have a different physical address of the form a.0.0.0 that the sources will read out and use as their physical address.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl CEC_DQEVENT

Name

CEC_DQEVENT - Dequeue a CEC event

Synopsis

int **ioctl**(int *fd*, *CEC_DQEVENT*, struct *cec_event* *argp)

Arguments

fd File descriptor returned by *open()* .

argp

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

CEC devices can send asynchronous events. These can be retrieved by calling *CEC_DQEVENT()*. If the file descriptor is in non-blocking mode and no event is pending, then it will return -1 and set errno to the EAGAIN error code.

The internal event queues are per-filehandle and per-event type. If there is no more room in a queue then the last event is overwritten with the new one. This means that intermediate results can be thrown away but that the latest event is always available. This also means that is it possible to read two successive events that have the same value (e.g. two *CEC_EVENT_STATE_CHANGE* events with the same state). In that case the intermediate state changes were lost but it is guaranteed that the state did change in between the two events.

cec_event_state_change

Table 1.223: struct cec_event_state_change

__u16	phys_addr	**The current physical address. This is CEC_PHYS_ADDR_INVALID if no** valid physical address is set.
__u16	log_addr_mask	The current set of claimed logical addresses. This is 0 if no logical addresses are claimed or if phys_addr is CEC_PHYS_ADDR_INVALID. If bit 15 is set (1 << CEC_LOG_ADDR_UNREGISTERED) then this device has the unregistered logical address. In that case all other bits are 0.

cec_event_lost_msgs

Table 1.224: struct cec_event_lost_msgs

__u32	lost_msgs	Set to the number of lost messages since the filehandle was opened or since the last time this event was dequeued for this filehandle. The messages lost are the oldest messages. So when a new message arrives and there is no more room, then the oldest message is discarded to make room for the new one. The internal size of the message queue guarantees that all messages received in the last two seconds will be stored. Since messages should be replied to within a second according to the CEC specification, this is more than enough.

cec_event

Table 1.225: struct cec_event

__u64	ts	Timestamp of the event in ns. The timestamp has been taken from the CLOCK_MONOTONIC clock. To access the same clock from userspace use clock_gettime().	
__u32	event	The CEC event type, see *CEC Events Types* .	
__u32	flags	Event flags, see *CEC Event Flags* .	
union	(anonymous)		
	struct cec_event_state_change	state_change	The new adapter state as sent by the *CEC_EVENT_STATE_CHANGE* event.
	struct cec_event_lost_msgs	lost_msgs	The number of lost messages as sent by the *CEC_EVENT_LOST_MSGS* event.

Table 1.226: CEC Events Types

CEC_EVENT_STATE_CHANGE	1	Generated when the CEC Adapter's state changes. When open() is called an initial event will be generated for that file-handle with the CEC Adapter's state at that time.
CEC_EVENT_LOST_MSGS	2	Generated if one or more CEC messages were lost because the application didn't dequeue CEC messages fast enough.

Table 1.227: CEC Event Flags

CEC_EVENT_FL_INITIAL_VALUE	1	Set for the initial events that are generated when the device is opened. See the table above for which events do this. This allows applications to learn the initial state of the CEC adapter at open() time.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls CEC_G_MODE and CEC_S_MODE

CEC_G_MODE, CEC_S_MODE - Get or set exclusive use of the CEC adapter

Synopsis

int **ioctl**(int *fd*, *CEC_G_MODE*, __u32 **argp*)

int **ioctl**(int *fd*, *CEC_S_MODE*, __u32 **argp*)

Arguments

fd File descriptor returned by *open()*.

argp Pointer to CEC mode.

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

By default any filehandle can use *ioctls CEC_RECEIVE and CEC_TRANSMIT* , but in order to prevent applications from stepping on each others toes it must be possible to obtain exclusive access to the CEC adapter. This ioctl sets the filehandle to initiator and/or follower mode which can be exclusive depending on the chosen mode. The initiator is the filehandle that is used to initiate messages, i.e. it commands other CEC devices. The follower is the filehandle that receives messages sent to the CEC adapter and processes them. The same filehandle can be both initiator and follower, or this role can be taken by two different filehandles.

When a CEC message is received, then the CEC framework will decide how it will be processed. If the message is a reply to an earlier transmitted message, then the reply is sent back to the filehandle that is waiting for it. In addition the CEC framework will process it.

If the message is not a reply, then the CEC framework will process it first. If there is no follower, then the message is just discarded and a feature abort is sent back to the initiator if the framework couldn't process it. If there is a follower, then the message is passed on to the follower who will use *ioctl CEC_RECEIVE* to dequeue the new message. The framework expects the follower to make the right decisions.

The CEC framework will process core messages unless requested otherwise by the follower. The follower can enable the passthrough mode. In that case, the CEC framework will pass on most core messages without processing them and the follower will have to implement those messages. There are some messages that the core will always process, regardless of the passthrough mode. See *Core Message Processing* for details.

If there is no initiator, then any CEC filehandle can use *ioctl CEC_TRANSMIT* . If there is an exclusive initiator then only that initiator can call *ioctls CEC_RECEIVE and CEC_TRANSMIT* . The follower can of course always call *ioctl CEC_TRANSMIT* .

Available initiator modes are:

Table 1.228: Initiator Modes

`CEC_MODE_NO_INITIATOR`	0x0	This is not an initiator, i.e. it cannot transmit CEC messages or make any other changes to the CEC adapter.
`CEC_MODE_INITIATOR`	0x1	This is an initiator (the default when the device is opened) and it can transmit CEC messages and make changes to the CEC adapter, unless there is an exclusive initiator.
`CEC_MODE_EXCL_INITIATOR`	0x2	This is an exclusive initiator and this file descriptor is the only one that can transmit CEC messages and make changes to the CEC adapter. If someone else is already the exclusive initiator then an attempt to become one will return the EBUSY error code error.

Available follower modes are:

Table 1.229: Follower Modes

CEC_MODE_NO_FOLLOWER	0x00	This is not a follower (the default when the device is opened).
CEC_MODE_FOLLOWER	0x10	This is a follower and it will receive CEC messages unless there is an exclusive follower. You cannot become a follower if *CEC_CAP_TRANSMIT* is not set or if *CEC_MODE_NO_INITIATOR* was specified, the EINVAL error code is returned in that case.
CEC_MODE_EXCL_FOLLOWER	0x20	This is an exclusive follower and only this file descriptor will receive CEC messages for processing. If someone else is already the exclusive follower then an attempt to become one will return the EBUSY error code. You cannot become a follower if *CEC_CAP_TRANSMIT* is not set or if *CEC_MODE_NO_INITIATOR* was specified, the EINVAL error code is returned in that case.
CEC_MODE_EXCL_FOLLOWER_PASSTHRU	0x30	This is an exclusive follower and only this file descriptor will receive CEC messages for processing. In addition it will put the CEC device into passthrough mode, allowing the exclusive follower to handle most core messages instead of relying on the CEC framework for that. If someone else is already the exclusive follower then an attempt to become one will return the EBUSY error code. You cannot become a follower if *CEC_CAP_TRANSMIT* is not set or if *CEC_MODE_NO_INITIATOR* was specified, the EINVAL error code is returned in that case.
CEC_MODE_MONITOR	0xe0	Put the file descriptor into monitor mode. Can only be used in combination with *CEC_MODE_NO_INITIATOR* , otherwise EINVAL error code will be returned. In monitor mode all messages this CEC device transmits and all messages it receives (both broadcast messages and directed messages for one its logical addresses) will be reported. This is very useful for debugging. This is only allowed if the process has the CAP_NET_ADMIN capability. If that is not set, then the EPERM error code is returned.
CEC_MODE_MONITOR_ALL	0xf0	Put the file descriptor into 'monitor all' mode. Can only be used in combination with *CEC_MODE_NO_INITIATOR* , otherwise the EINVAL error code will be returned. In 'monitor all' mode all messages this CEC device transmits and all messages it receives, including directed messages for other CEC devices will be reported. This is very useful for debugging, but not all devices support this. This mode requires that the *CEC_CAP_MONITOR_ALL* capability is set, otherwise the EINVAL error code is returned. This is only allowed if the process has the CAP_NET_ADMIN capability. If that is not set, then the EPERM error code is returned.

Core message processing details:

Table 1.230: Core Message Processing

CEC_MSG_GET_CEC_VERSION	When in passthrough mode this message has to be handled by userspace, otherwise the core will return the CEC version that was set with *ioctl CEC_ADAP_S_LOG_ADDRS* .
CEC_MSG_GIVE_DEVICE_VENDOR_ID	When in passthrough mode this message has to be handled by userspace, otherwise the core will return the vendor ID that was set with *ioctl CEC_ADAP_S_LOG_ADDRS* .
CEC_MSG_ABORT	When in passthrough mode this message has to be handled by userspace, otherwise the core will return a feature refused message as per the specification.
CEC_MSG_GIVE_PHYSICAL_ADDR	When in passthrough mode this message has to be handled by userspace, otherwise the core will report the current physical address.
CEC_MSG_GIVE_OSD_NAME	When in passthrough mode this message has to be handled by userspace, otherwise the core will report the current OSD name as was set with *ioctl CEC_ADAP_S_LOG_ADDRS* .
CEC_MSG_GIVE_FEATURES	When in passthrough mode this message has to be handled by userspace, otherwise the core will report the current features as was set with *ioctl CEC_ADAP_S_LOG_ADDRS* or the message is ignored if the CEC version was older than 2.0.
CEC_MSG_USER_CONTROL_PRESSED	If *CEC_CAP_RC* is set, then generate a remote control key press. This message is always passed on to userspace.
CEC_MSG_USER_CONTROL_RELEASED	If *CEC_CAP_RC* is set, then generate a remote control key release. This message is always passed on to userspace.
CEC_MSG_REPORT_PHYSICAL_ADDR	The CEC framework will make note of the reported physical address and then just pass the message on to userspace.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls CEC_RECEIVE and CEC_TRANSMIT

Name

CEC_RECEIVE, CEC_TRANSMIT - Receive or transmit a CEC message

Synopsis

int **ioctl**(int *fd*, *CEC_RECEIVE*, struct *cec_msg* *argp)

int **ioctl**(int *fd*, *CEC_TRANSMIT*, struct *cec_msg* *argp)

Arguments

fd File descriptor returned by *open()*.

argp Pointer to struct cec_msg.

Description

Note:

This documents the proposed CEC API. This API is not yet finalized and is currently only available as a staging kernel module.

To receive a CEC message the application has to fill in the `timeout` field of struct *cec_msg* and pass it to *ioctl CEC_RECEIVE* . If the file descriptor is in non-blocking mode and there are no received messages pending, then it will return -1 and set errno to the `EAGAIN` error code. If the file descriptor is in blocking mode and `timeout` is non-zero and no message arrived within `timeout` milliseconds, then it will return -1 and set errno to the `ETIMEDOUT` error code.

A received message can be:

1. a message received from another CEC device (the sequence field will be 0).
2. the result of an earlier non-blocking transmit (the sequence field will be non-zero).

To send a CEC message the application has to fill in the struct :c:type:‘ cec_msg‘ and pass it to *ioctl CEC_TRANSMIT* . The *ioctl CEC_TRANSMIT* is only available if `CEC_CAP_TRANSMIT` is set. If there is no more room in the transmit queue, then it will return -1 and set errno to the `EBUSY` error code. The transmit queue has enough room for 18 messages (about 1 second worth of 2-byte messages). Note that the CEC kernel framework will also reply to core messages (see :ref:cec-core-processing), so it is not a good idea to fully fill up the transmit queue.

If the file descriptor is in non-blocking mode then the transmit will return 0 and the result of the transmit will be available via *ioctl CEC_RECEIVE* once the transmit has finished (including waiting for a reply, if requested).

The sequence field is filled in for every transmit and this can be checked against the received messages to find the corresponding transmit result.

cec_msg

Table 1.231: struct cec_msg

__u64	`tx_ts`	Timestamp in ns of when the last byte of the message was transmitted. The timestamp has been taken from the `CLOCK_MONOTONIC` clock. To access the same clock from userspace use `clock_gettime()`.
__u64	`rx_ts`	Timestamp in ns of when the last byte of the message was received. The timestamp has been taken from the `CLOCK_MONOTONIC` clock. To access the same clock from userspace use `clock_gettime()`.
__u32	`len`	The length of the message. For *ioctl CEC_TRANSMIT* this is filled in by the application. The driver will fill this in for *ioctl CEC_RECEIVE* . For *ioctl CEC_TRANSMIT* it will be filled in by the driver with the length of the reply message if `reply` was set.
__u32	`timeout`	The timeout in milliseconds. This is the time the device will wait for a message to be received before timing out. If it is set to 0, then it will wait indefinitely when it is called by *ioctl CEC_RECEIVE* . If it is 0 and it is called by *ioctl CEC_TRANSMIT* , then it will be replaced by 1000 if the `reply` is non-zero or ignored if `reply` is 0.

Continued on next page

Table 1.231 – continued from previous page

__u32	sequence	A non-zero sequence number is automatically assigned by the CEC framework for all transmitted messages. It is used by the CEC framework when it queues the transmit result (when transmit was called in non-blocking mode). This allows the application to associate the received message with the original transmit.
__u32	flags	Flags. See *Flags for struct cec_msg* for a list of available flags.
__u8	tx_status	The status bits of the transmitted message. See *CEC Transmit Status* for the possible status values. It is 0 if this messages was received, not transmitted.
__u8	msg[16]	The message payload. For *ioctl CEC_TRANSMIT* this is filled in by the application. The driver will fill this in for *ioctl CEC_RECEIVE* . For *ioctl CEC_TRANSMIT* it will be filled in by the driver with the payload of the reply message if timeout was set.
__u8	reply	Wait until this message is replied. If reply is 0 and the timeout is 0, then don't wait for a reply but return after transmitting the message. Ignored by *ioctl CEC_RECEIVE* . The case where reply is 0 (this is the opcode for the Feature Abort message) and timeout is non-zero is specifically allowed to make it possible to send a message and wait up to timeout milliseconds for a Feature Abort reply. In this case rx_status will either be set to *CEC_RX_STATUS_TIMEOUT* or *CEC_RX_STATUS_FEATURE_ABORT* . If the transmitter message is CEC_MSG_INITIATE_ARC then the reply values CEC_MSG_REPORT_ARC_INITIATED and CEC_MSG_REPORT_ARC_TERMINATED are processed differently: either value will match both possible replies. The reason is that the CEC_MSG_INITIATE_ARC message is the only CEC message that has two possible replies other than Feature Abort. The reply field will be updated with the actual reply so that it is synchronized with the contents of the received message.
__u8	rx_status	The status bits of the received message. See *CEC Receive Status* for the possible status values. It is 0 if this message was transmitted, not received, unless this is the reply to a transmitted message. In that case both rx_status and tx_status are set.
__u8	tx_status	The status bits of the transmitted message. See *CEC Transmit Status* for the possible status values. It is 0 if this messages was received, not transmitted.
__u8	tx_arb_lost_cnt	A counter of the number of transmit attempts that resulted in the Arbitration Lost error. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the *CEC_TX_STATUS_ARB_LOST* status bit is set.
__u8	tx_nack_cnt	A counter of the number of transmit attempts that resulted in the Not Acknowledged error. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the *CEC_TX_STATUS_NACK* status bit is set.
__u8	tx_low_drive_cnt	A counter of the number of transmit attempts that resulted in the Arbitration Lost error. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the *CEC_TX_STATUS_LOW_DRIVE* status bit is set.
__u8	tx_error_cnt	A counter of the number of transmit errors other than Arbitration Lost or Not Acknowledged. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the *CEC_TX_STATUS_ERROR* status bit is set.

Table 1.232: Flags for struct cec_msg

CEC_MSG_FL_REPLY_TO_FOLLOWERS	1	If this transmit expects a reply, then by default that reply is only sent to the filehandle that called *ioctl CEC_TRANSMIT* . If this flag is set, then the reply is also sent to all followers, if any. If the filehandle that called *ioctl CEC_TRANSMIT* is also a follower, then that filehandle will receive the reply twice: once as the result of the *ioctl CEC_TRANSMIT* , and once via *ioctl CEC_RECEIVE* .

Table 1.233: CEC Transmit Status

CEC_TX_STATUS_OK	0x01	The message was transmitted successfully. This is mutually exclusive with *CEC_TX_STATUS_MAX_RETRIES* . Other bits can still be set if earlier attempts met with failure before the transmit was eventually successful.
CEC_TX_STATUS_ARB_LOST	0x02	CEC line arbitration was lost.
CEC_TX_STATUS_NACK	0x04	Message was not acknowledged.
CEC_TX_STATUS_LOW_DRIVE	0x08	Low drive was detected on the CEC bus. This indicates that a follower detected an error on the bus and requests a retransmission.
CEC_TX_STATUS_ERROR	0x10	Some error occurred. This is used for any errors that do not fit the previous two, either because the hardware could not tell which error occurred, or because the hardware tested for other conditions besides those two.
CEC_TX_STATUS_MAX_RETRIES	0x20	The transmit failed after one or more retries. This status bit is mutually exclusive with *CEC_TX_STATUS_OK* . Other bits can still be set to explain which failures were seen.

Table 1.234: CEC Receive Status

CEC_RX_STATUS_OK	0x01	The message was received successfully.
CEC_RX_STATUS_TIMEOUT	0x02	The reply to an earlier transmitted message timed out.
CEC_RX_STATUS_FEATURE_ABORT	0x04	The message was received successfully but the reply was CEC_MSG_FEATURE_ABORT. This status is only set if this message was the reply to an earlier transmitted message.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

1.6.3 CEC Header File

cec.h

```
/*
 * cec - HDMI Consumer Electronics Control public header
 *
 * Copyright 2016 Cisco Systems, Inc. and/or its affiliates. All rights reserved.
```

```
 *
 * This program is free software; you may redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; version 2 of the License.
 *
 * Alternatively you can redistribute this file under the terms of the
 * BSD license as stated below:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 * 3. The names of its contributors may not be used to endorse or promote
 *    products derived from this software without specific prior written
 *    permission.
 *
 * THE SOFTWARE IS PROVIDED ``AS IS'', WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
 * BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */

#ifndef _CEC_UAPI_H
#define _CEC_UAPI_H

#include <linux/types.h>
#include <linux/string.h>

#define CEC_MAX_MSG_SIZE        16

/**
 * struct  cec_msg
 - CEC message structure.
 * @tx_ts:      Timestamp in nanoseconds using CLOCK_MONOTONIC. Set by the
 *              driver when the message transmission has finished.
 * @rx_ts:      Timestamp in nanoseconds using CLOCK_MONOTONIC. Set by the
 *              driver when the message was received.
 * @len:        Length in bytes of the message.
 * @timeout:    The timeout (in ms) that is used to timeout CEC_RECEIVE.
 *              Set to 0 if you want to wait forever. This timeout can also be
 *              used with  CEC_TRANSMIT  as the timeout for waiting for a reply.
 *              If 0, then it will use a 1 second timeout instead of waiting
 *              forever as is done with CEC_RECEIVE.
 * @sequence:   The framework assigns a sequence number to messages that are
 *              sent. This can be used to track replies to previously sent
 *              messages.
 * @flags:      Set to 0.
 * @msg:        The message payload.
 * @reply:      This field is ignored with  CEC_RECEIVE  and is only used by
```

```
 *              CEC_TRANSMIT. If non-zero, then wait for a reply with this
 *              opcode. Set to CEC_MSG_FEATURE_ABORT if you want to wait for
 *              a possible ABORT reply. If there was an error when sending the
 *              msg or FeatureAbort was returned, then reply is set to 0.
 *              If reply is non-zero upon return, then len/msg are set to
 *              the received message.
 *              If reply is zero upon return and status has the
 *              CEC_TX_STATUS_FEATURE_ABORT bit set, then len/msg are set to
 *              the received feature abort message.
 *              If reply is zero upon return and status has the
 *               CEC_TX_STATUS_MAX_RETRIES  bit set, then no reply was seen at
 *              all. If reply is non-zero for  CEC_TRANSMIT  and the message is a
 *              broadcast, then -EINVAL is returned.
 *              if reply is non-zero, then timeout is set to 1000 (the required
 *              maximum response time).
 * @rx_status:  The message receive status bits. Set by the driver.
 * @tx_status:  The message transmit status bits. Set by the driver.
 * @tx_arb_lost_cnt: The number of `Arbitration Lost' events. Set by the driver.
 * @tx_nack_cnt: The number of `Not Acknowledged' events. Set by the driver.
 * @tx_low_drive_cnt: The number of `Low Drive Detected' events. Set by the
 *              driver.
 * @tx_error_cnt: The number of `Error' events. Set by the driver.
 */
struct  cec_msg
 {
        __u64 tx_ts;
        __u64 rx_ts;
        __u32 len;
        __u32 timeout;
        __u32 sequence;
        __u32 flags;
        __u8 msg[CEC_MAX_MSG_SIZE];
        __u8 reply;
        __u8 rx_status;
        __u8 tx_status;
        __u8 tx_arb_lost_cnt;
        __u8 tx_nack_cnt;
        __u8 tx_low_drive_cnt;
        __u8 tx_error_cnt;
};

/**
 * cec_msg_initiator - return the initiator's logical address.
 * @msg:        the message structure
 */
static inline __u8 cec_msg_initiator(const struct  cec_msg
 *msg)
{
        return msg->msg[0] >> 4;
}

/**
 * cec_msg_destination - return the destination's logical address.
 * @msg:        the message structure
 */
static inline __u8 cec_msg_destination(const struct  cec_msg
 *msg)
{
```

```
        return msg->msg[0] & 0xf;
}

/**
 * cec_msg_opcode - return the opcode of the message, -1 for poll
 * @msg:        the message structure
 */
static inline int cec_msg_opcode(const struct  cec_msg
 *msg)
{
        return msg->len > 1 ? msg->msg[1] : -1;
}

/**
 * cec_msg_is_broadcast - return true if this is a broadcast message.
 * @msg:        the message structure
 */
static inline int cec_msg_is_broadcast(const struct  cec_msg
 *msg)
{
        return (msg->msg[0] & 0xf) == 0xf;
}

/**
 * cec_msg_init - initialize the message structure.
 * @msg:        the message structure
 * @initiator:  the logical address of the initiator
 * @destination:the logical address of the destination (0xf for broadcast)
 *
 * The whole structure is zeroed, the len field is set to 1 (i.e. a poll
 * message) and the initiator and destination are filled in.
 */
static inline void cec_msg_init(struct  cec_msg
 *msg,
                                __u8 initiator, __u8 destination)
{
        memset(msg, 0, sizeof(*msg));
        msg->msg[0] = (initiator << 4) | destination;
        msg->len = 1;
}

/**
 * cec_msg_set_reply_to - fill in destination/initiator in a reply message.
 * @msg:        the message structure for the reply
 * @orig:       the original message structure
 *
 * Set the msg destination to the orig initiator and the msg initiator to the
 * orig destination. Note that msg and orig may be the same pointer, in which
 * case the change is done in place.
 */
static inline void cec_msg_set_reply_to(struct  cec_msg
 *msg,
                                        struct  cec_msg
 *orig)
{
        /* The destination becomes the initiator and vice versa */
        msg->msg[0] = (cec_msg_destination(orig) << 4) |
                      cec_msg_initiator(orig);
```

```
        msg->reply = msg->timeout = 0;
}

/* cec_msg flags field */
#define  CEC_MSG_FL_REPLY_TO_FOLLOWERS    (1 << 0)

/* cec_msg tx/rx_status field */
#define  CEC_TX_STATUS_OK                 (1 << 0)
#define  CEC_TX_STATUS_ARB_LOST           (1 << 1)
#define  CEC_TX_STATUS_NACK               (1 << 2)
#define  CEC_TX_STATUS_LOW_DRIVE          (1 << 3)
#define  CEC_TX_STATUS_ERROR              (1 << 4)
#define  CEC_TX_STATUS_MAX_RETRIES        (1 << 5)

#define  CEC_RX_STATUS_OK                 (1 << 0)
#define  CEC_RX_STATUS_TIMEOUT            (1 << 1)
#define  CEC_RX_STATUS_FEATURE_ABORT      (1 << 2)

static inline int cec_msg_status_is_ok(const struct  cec_msg
 *msg)
{
        if (msg->tx_status && !(msg->tx_status &  CEC_TX_STATUS_OK ))
                return 0;
        if (msg->rx_status && !(msg->rx_status &  CEC_RX_STATUS_OK ))
                return 0;
        if (!msg->tx_status && !msg->rx_status)
                return 0;
        return !(msg->rx_status &  CEC_RX_STATUS_FEATURE_ABORT );
}

#define CEC_LOG_ADDR_INVALID            0xff
#define CEC_PHYS_ADDR_INVALID           0xffff

/*
 * The maximum number of logical addresses one device can be assigned to.
 * The CEC 2.0 spec allows for only 2 logical addresses at the moment. The
 * Analog Devices CEC hardware supports 3. So let's go wild and go for 4.
 */
#define CEC_MAX_LOG_ADDRS 4

/* The logical addresses defined by CEC 2.0 */
#define CEC_LOG_ADDR_TV                 0
#define CEC_LOG_ADDR_RECORD_1           1
#define CEC_LOG_ADDR_RECORD_2           2
#define CEC_LOG_ADDR_TUNER_1            3
#define CEC_LOG_ADDR_PLAYBACK_1         4
#define CEC_LOG_ADDR_AUDIOSYSTEM        5
#define CEC_LOG_ADDR_TUNER_2            6
#define CEC_LOG_ADDR_TUNER_3            7
#define CEC_LOG_ADDR_PLAYBACK_2         8
#define CEC_LOG_ADDR_RECORD_3           9
#define CEC_LOG_ADDR_TUNER_4            10
#define CEC_LOG_ADDR_PLAYBACK_3         11
#define CEC_LOG_ADDR_BACKUP_1           12
#define CEC_LOG_ADDR_BACKUP_2           13
#define CEC_LOG_ADDR_SPECIFIC           14
#define CEC_LOG_ADDR_UNREGISTERED       15 /* as initiator address */
#define CEC_LOG_ADDR_BROADCAST          15 /* ad destination address */
```

```
/* The logical address types that the CEC device wants to claim */
#define  CEC_LOG_ADDR_TYPE_TV             0
#define  CEC_LOG_ADDR_TYPE_RECORD         1
#define  CEC_LOG_ADDR_TYPE_TUNER          2
#define  CEC_LOG_ADDR_TYPE_PLAYBACK       3
#define  CEC_LOG_ADDR_TYPE_AUDIOSYSTEM    4
#define  CEC_LOG_ADDR_TYPE_SPECIFIC       5
#define  CEC_LOG_ADDR_TYPE_UNREGISTERED   6
/*
 * Switches should use UNREGISTERED.
 * Processors should use SPECIFIC.
 */

#define CEC_LOG_ADDR_MASK_TV            (1 << CEC_LOG_ADDR_TV)
#define CEC_LOG_ADDR_MASK_RECORD        ((1 << CEC_LOG_ADDR_RECORD_1) | \
                                         (1 << CEC_LOG_ADDR_RECORD_2) | \
                                         (1 << CEC_LOG_ADDR_RECORD_3))
#define CEC_LOG_ADDR_MASK_TUNER         ((1 << CEC_LOG_ADDR_TUNER_1) | \
                                         (1 << CEC_LOG_ADDR_TUNER_2) | \
                                         (1 << CEC_LOG_ADDR_TUNER_3) | \
                                         (1 << CEC_LOG_ADDR_TUNER_4))
#define CEC_LOG_ADDR_MASK_PLAYBACK      ((1 << CEC_LOG_ADDR_PLAYBACK_1) | \
                                         (1 << CEC_LOG_ADDR_PLAYBACK_2) | \
                                         (1 << CEC_LOG_ADDR_PLAYBACK_3))
#define CEC_LOG_ADDR_MASK_AUDIOSYSTEM   (1 << CEC_LOG_ADDR_AUDIOSYSTEM)
#define CEC_LOG_ADDR_MASK_BACKUP        ((1 << CEC_LOG_ADDR_BACKUP_1) | \
                                         (1 << CEC_LOG_ADDR_BACKUP_2))
#define CEC_LOG_ADDR_MASK_SPECIFIC      (1 << CEC_LOG_ADDR_SPECIFIC)
#define CEC_LOG_ADDR_MASK_UNREGISTERED  (1 << CEC_LOG_ADDR_UNREGISTERED)

static inline int cec_has_tv(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_TV;
}

static inline int cec_has_record(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_RECORD;
}

static inline int cec_has_tuner(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_TUNER;
}

static inline int cec_has_playback(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_PLAYBACK;
}

static inline int cec_has_audiosystem(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_AUDIOSYSTEM;
}

static inline int cec_has_backup(__u16 log_addr_mask)
{
```

```
        return log_addr_mask & CEC_LOG_ADDR_MASK_BACKUP;
}

static inline int cec_has_specific(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_SPECIFIC;
}

static inline int cec_is_unregistered(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_UNREGISTERED;
}

static inline int cec_is_unconfigured(__u16 log_addr_mask)
{
        return log_addr_mask == 0;
}

/*
 * Use this if there is no vendor ID (CEC_G_VENDOR_ID) or if the vendor ID
 * should be disabled (CEC_S_VENDOR_ID)
 */
#define CEC_VENDOR_ID_NONE              0xffffffff

/* The message handling modes */
/* Modes for initiator */
#define  CEC_MODE_NO_INITIATOR            (0x0 << 0)
#define  CEC_MODE_INITIATOR               (0x1 << 0)
#define  CEC_MODE_EXCL_INITIATOR          (0x2 << 0)
#define CEC_MODE_INITIATOR_MSK          0x0f

/* Modes for follower */
#define  CEC_MODE_NO_FOLLOWER             (0x0 << 4)
#define  CEC_MODE_FOLLOWER                (0x1 << 4)
#define  CEC_MODE_EXCL_FOLLOWER           (0x2 << 4)
#define  CEC_MODE_EXCL_FOLLOWER_PASSTHRU  (0x3 << 4)
#define  CEC_MODE_MONITOR                 (0xe << 4)
#define  CEC_MODE_MONITOR_ALL             (0xf << 4)
#define CEC_MODE_FOLLOWER_MSK           0xf0

/* Userspace has to configure the physical address */
#define  CEC_CAP_PHYS_ADDR        (1 << 0)
/* Userspace has to configure the logical addresses */
#define  CEC_CAP_LOG_ADDRS        (1 << 1)
/* Userspace can transmit messages (and thus become follower as well) */
#define  CEC_CAP_TRANSMIT         (1 << 2)
/*
 * Passthrough all messages instead of processing them.
 */
#define  CEC_CAP_PASSTHROUGH      (1 << 3)
/* Supports remote control */
#define  CEC_CAP_RC               (1 << 4)
/* Hardware can monitor all messages, not just directed and broadcast. */
#define  CEC_CAP_MONITOR_ALL      (1 << 5)

/**
 * struct  cec_caps
 - CEC capabilities structure.
```

```
 * @driver: name of the CEC device driver.
 * @name: name of the CEC device. @driver + @name must be unique.
 * @available_log_addrs: number of available logical addresses.
 * @capabilities: capabilities of the CEC adapter.
 * @version: version of the CEC adapter framework.
 */
struct  cec_caps
 {
        char driver[32];
        char name[32];
        __u32 available_log_addrs;
        __u32 capabilities;
        __u32 version;
};

/**
 * struct  cec_log_addrs
 - CEC logical addresses structure.
 * @log_addr: the claimed logical addresses. Set by the driver.
 * @log_addr_mask: current logical address mask. Set by the driver.
 * @cec_version: the CEC version that the adapter should implement. Set by the
 *      caller.
 * @num_log_addrs: how many logical addresses should be claimed. Set by the
 *      caller.
 * @vendor_id: the vendor ID of the device. Set by the caller.
 * @flags: flags.
 * @osd_name: the OSD name of the device. Set by the caller.
 * @primary_device_type: the primary device type for each logical address.
 *      Set by the caller.
 * @log_addr_type: the logical address types. Set by the caller.
 * @all_device_types: CEC 2.0: all device types represented by the logical
 *      address. Set by the caller.
 * @features:   CEC 2.0: The logical address features. Set by the caller.
 */
struct  cec_log_addrs
 {
        __u8 log_addr[CEC_MAX_LOG_ADDRS];
        __u16 log_addr_mask;
        __u8 cec_version;
        __u8 num_log_addrs;
        __u32 vendor_id;
        __u32 flags;
        char osd_name[15];
        __u8 primary_device_type[CEC_MAX_LOG_ADDRS];
        __u8 log_addr_type[CEC_MAX_LOG_ADDRS];

        /* CEC 2.0 */
        __u8 all_device_types[CEC_MAX_LOG_ADDRS];
        __u8 features[CEC_MAX_LOG_ADDRS][12];
};

/* Allow a fallback to unregistered */
#define  CEC_LOG_ADDRS_FL_ALLOW_UNREG_FALLBACK    (1 << 0)
/* Passthrough RC messages to the input subsystem */
#define  CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU       (1 << 1)
/* CDC-Only device: supports only CDC messages */
#define  CEC_LOG_ADDRS_FL_CDC_ONLY                (1 << 2)
```

```
/* Events */

/* Event that occurs when the adapter state changes */
#define  CEC_EVENT_STATE_CHANGE           1
/*
 * This event is sent when messages are lost because the application
 * didn't empty the message queue in time
 */
#define  CEC_EVENT_LOST_MSGS              2

#define CEC_EVENT_FL_INITIAL_STATE      (1 << 0)

/**
 * struct  cec_event_state_change
 - used when the CEC adapter changes state.
 * @phys_addr: the current physical address
 * @log_addr_mask: the current logical address mask
 */
struct  cec_event_state_change
 {
        __u16 phys_addr;
        __u16 log_addr_mask;
};

/**
 * struct  cec_event_lost_msgs
 - tells you how many messages were lost due.
 * @lost_msgs: how many messages were lost.
 */
struct  cec_event_lost_msgs
 {
        __u32 lost_msgs;
};

/**
 * struct  cec_event
 - CEC event structure
 * @ts: the timestamp of when the event was sent.
 * @event: the event.
 * array.
 * @state_change: the event payload for CEC_EVENT_STATE_CHANGE.
 * @lost_msgs: the event payload for CEC_EVENT_LOST_MSGS.
 * @raw: array to pad the union.
 */
struct  cec_event
 {
        __u64 ts;
        __u32 event;
        __u32 flags;
        union {
                struct  cec_event_state_change
 state_change;
                struct  cec_event_lost_msgs
 lost_msgs;
                __u32 raw[16];
        };
};
```

```
/* ioctls */

/* Adapter capabilities */
#define  CEC_ADAP_G_CAPS          _IOWR(`a',  0, struct  cec_caps
)

/*
 * phys_addr is either 0 (if this is the CEC root device)
 * or a valid physical address obtained from the sink's EDID
 * as read by this CEC device (if this is a source device)
 * or a physical address obtained and modified from a sink
 * EDID and used for a sink CEC device.
 * If nothing is connected, then phys_addr is 0xffff.
 * See HDMI 1.4b, section 8.7 (Physical Address).
 *
 * The  CEC_ADAP_S_PHYS_ADDR  ioctl may not be available if that is handled
 * internally.
 */
#define  CEC_ADAP_G_PHYS_ADDR     _IOR(`a',  1, __u16)
#define  CEC_ADAP_S_PHYS_ADDR     _IOW(`a',  2, __u16)

/*
 * Configure the CEC adapter. It sets the device type and which
 * logical types it will try to claim. It will return which
 * logical addresses it could actually claim.
 * An error is returned if the adapter is disabled or if there
 * is no physical address assigned.
 */

#define  CEC_ADAP_G_LOG_ADDRS     _IOR(`a',  3, struct  cec_log_addrs
)
#define  CEC_ADAP_S_LOG_ADDRS     _IOWR(`a',  4, struct  cec_log_addrs
)

/* Transmit/receive a CEC command */
#define  CEC_TRANSMIT             _IOWR(`a',  5, struct  cec_msg
)
#define  CEC_RECEIVE              _IOWR(`a',  6, struct  cec_msg
)

/* Dequeue CEC events */
#define  CEC_DQEVENT              _IOWR(`a',  7, struct  cec_event
)

/*
 * Get and set the message handling mode for this filehandle.
 */
#define  CEC_G_MODE               _IOR(`a',  8, __u32)
#define  CEC_S_MODE               _IOW(`a',  9, __u32)

/*
 * The remainder of this header defines all CEC messages and operands.
 * The format matters since it the cec-ctl utility parses it to generate
 * code for implementing all these messages.
 *
 * Comments ending with `Feature' group messages for each feature.
 * If messages are part of multiple features, then the ``Has also''
 * comment is used to list the previously defined messages that are
```

```
 * supported by the feature.
 *
 * Before operands are defined a comment is added that gives the
 * name of the operand and in brackets the variable name of the
 * corresponding argument in the cec-funcs.h function.
 */

/* Messages */

/* One Touch Play Feature */
#define CEC_MSG_ACTIVE_SOURCE                           0x82
#define CEC_MSG_IMAGE_VIEW_ON                           0x04
#define CEC_MSG_TEXT_VIEW_ON                            0x0d

/* Routing Control Feature */

/*
 * Has also:
 *      CEC_MSG_ACTIVE_SOURCE
 */

#define CEC_MSG_INACTIVE_SOURCE                         0x9d
#define CEC_MSG_REQUEST_ACTIVE_SOURCE                   0x85
#define CEC_MSG_ROUTING_CHANGE                          0x80
#define CEC_MSG_ROUTING_INFORMATION                     0x81
#define CEC_MSG_SET_STREAM_PATH                         0x86

/* Standby Feature */
#define CEC_MSG_STANDBY                                 0x36

/* One Touch Record Feature */
#define CEC_MSG_RECORD_OFF                              0x0b
#define CEC_MSG_RECORD_ON                               0x09
/* Record Source Type Operand (rec_src_type) */
#define CEC_OP_RECORD_SRC_OWN                           1
#define CEC_OP_RECORD_SRC_DIGITAL                       2
#define CEC_OP_RECORD_SRC_ANALOG                        3
#define CEC_OP_RECORD_SRC_EXT_PLUG                      4
#define CEC_OP_RECORD_SRC_EXT_PHYS_ADDR                 5
/* Service Identification Method Operand (service_id_method) */
#define CEC_OP_SERVICE_ID_METHOD_BY_DIG_ID              0
#define CEC_OP_SERVICE_ID_METHOD_BY_CHANNEL             1
/* Digital Service Broadcast System Operand (dig_bcast_system) */
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_GEN        0x00
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_GEN        0x01
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_GEN         0x02
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_BS         0x08
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_CS         0x09
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_T          0x0a
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_CABLE      0x10
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_SAT        0x11
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_T          0x12
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_C           0x18
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_S           0x19
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_S2          0x1a
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_T           0x1b
/* Analogue Broadcast Type Operand (ana_bcast_type) */
#define CEC_OP_ANA_BCAST_TYPE_CABLE                     0
```

```
#define CEC_OP_ANA_BCAST_TYPE_SATELLITE                 1
#define CEC_OP_ANA_BCAST_TYPE_TERRESTRIAL               2
/* Broadcast System Operand (bcast_system) */
#define CEC_OP_BCAST_SYSTEM_PAL_BG                      0x00
#define CEC_OP_BCAST_SYSTEM_SECAM_LQ                    0x01 /* SECAM L' */
#define CEC_OP_BCAST_SYSTEM_PAL_M                       0x02
#define CEC_OP_BCAST_SYSTEM_NTSC_M                      0x03
#define CEC_OP_BCAST_SYSTEM_PAL_I                       0x04
#define CEC_OP_BCAST_SYSTEM_SECAM_DK                    0x05
#define CEC_OP_BCAST_SYSTEM_SECAM_BG                    0x06
#define CEC_OP_BCAST_SYSTEM_SECAM_L                     0x07
#define CEC_OP_BCAST_SYSTEM_PAL_DK                      0x08
#define CEC_OP_BCAST_SYSTEM_OTHER                       0x1f
/* Channel Number Format Operand (channel_number_fmt) */
#define CEC_OP_CHANNEL_NUMBER_FMT_1_PART                0x01
#define CEC_OP_CHANNEL_NUMBER_FMT_2_PART                0x02

#define CEC_MSG_RECORD_STATUS                           0x0a
/* Record Status Operand (rec_status) */
#define CEC_OP_RECORD_STATUS_CUR_SRC                    0x01
#define CEC_OP_RECORD_STATUS_DIG_SERVICE                0x02
#define CEC_OP_RECORD_STATUS_ANA_SERVICE                0x03
#define CEC_OP_RECORD_STATUS_EXT_INPUT                  0x04
#define CEC_OP_RECORD_STATUS_NO_DIG_SERVICE             0x05
#define CEC_OP_RECORD_STATUS_NO_ANA_SERVICE             0x06
#define CEC_OP_RECORD_STATUS_NO_SERVICE                 0x07
#define CEC_OP_RECORD_STATUS_INVALID_EXT_PLUG           0x09
#define CEC_OP_RECORD_STATUS_INVALID_EXT_PHYS_ADDR      0x0a
#define CEC_OP_RECORD_STATUS_UNSUP_CA                   0x0b
#define CEC_OP_RECORD_STATUS_NO_CA_ENTITLEMENTS         0x0c
#define CEC_OP_RECORD_STATUS_CANT_COPY_SRC              0x0d
#define CEC_OP_RECORD_STATUS_NO_MORE_COPIES             0x0e
#define CEC_OP_RECORD_STATUS_NO_MEDIA                   0x10
#define CEC_OP_RECORD_STATUS_PLAYING                    0x11
#define CEC_OP_RECORD_STATUS_ALREADY_RECORDING          0x12
#define CEC_OP_RECORD_STATUS_MEDIA_PROT                 0x13
#define CEC_OP_RECORD_STATUS_NO_SIGNAL                  0x14
#define CEC_OP_RECORD_STATUS_MEDIA_PROBLEM              0x15
#define CEC_OP_RECORD_STATUS_NO_SPACE                   0x16
#define CEC_OP_RECORD_STATUS_PARENTAL_LOCK              0x17
#define CEC_OP_RECORD_STATUS_TERMINATED_OK              0x1a
#define CEC_OP_RECORD_STATUS_ALREADY_TERM               0x1b
#define CEC_OP_RECORD_STATUS_OTHER                      0x1f

#define CEC_MSG_RECORD_TV_SCREEN                        0x0f

/* Timer Programming Feature */
#define CEC_MSG_CLEAR_ANALOGUE_TIMER                    0x33
/* Recording Sequence Operand (recording_seq) */
#define CEC_OP_REC_SEQ_SUNDAY                           0x01
#define CEC_OP_REC_SEQ_MONDAY                           0x02
#define CEC_OP_REC_SEQ_TUESDAY                          0x04
#define CEC_OP_REC_SEQ_WEDNESDAY                        0x08
#define CEC_OP_REC_SEQ_THURSDAY                         0x10
#define CEC_OP_REC_SEQ_FRIDAY                           0x20
#define CEC_OP_REC_SEQ_SATERDAY                         0x40
#define CEC_OP_REC_SEQ_ONCE_ONLY                        0x00
```

```
#define CEC_MSG_CLEAR_DIGITAL_TIMER                     0x99

#define CEC_MSG_CLEAR_EXT_TIMER                         0xa1
/* External Source Specifier Operand (ext_src_spec) */
#define CEC_OP_EXT_SRC_PLUG                             0x04
#define CEC_OP_EXT_SRC_PHYS_ADDR                        0x05

#define CEC_MSG_SET_ANALOGUE_TIMER                      0x34
#define CEC_MSG_SET_DIGITAL_TIMER                       0x97
#define CEC_MSG_SET_EXT_TIMER                           0xa2

#define CEC_MSG_SET_TIMER_PROGRAM_TITLE                 0x67
#define CEC_MSG_TIMER_CLEARED_STATUS                    0x43
/* Timer Cleared Status Data Operand (timer_cleared_status) */
#define CEC_OP_TIMER_CLR_STAT_RECORDING                 0x00
#define CEC_OP_TIMER_CLR_STAT_NO_MATCHING               0x01
#define CEC_OP_TIMER_CLR_STAT_NO_INFO                   0x02
#define CEC_OP_TIMER_CLR_STAT_CLEARED                   0x80

#define CEC_MSG_TIMER_STATUS                            0x35
/* Timer Overlap Warning Operand (timer_overlap_warning) */
#define CEC_OP_TIMER_OVERLAP_WARNING_NO_OVERLAP         0
#define CEC_OP_TIMER_OVERLAP_WARNING_OVERLAP            1
/* Media Info Operand (media_info) */
#define CEC_OP_MEDIA_INFO_UNPROT_MEDIA                  0
#define CEC_OP_MEDIA_INFO_PROT_MEDIA                    1
#define CEC_OP_MEDIA_INFO_NO_MEDIA                      2
/* Programmed Indicator Operand (prog_indicator) */
#define CEC_OP_PROG_IND_NOT_PROGRAMMED                  0
#define CEC_OP_PROG_IND_PROGRAMMED                      1
/* Programmed Info Operand (prog_info) */
#define CEC_OP_PROG_INFO_ENOUGH_SPACE                   0x08
#define CEC_OP_PROG_INFO_NOT_ENOUGH_SPACE               0x09
#define CEC_OP_PROG_INFO_MIGHT_NOT_BE_ENOUGH_SPACE      0x0b
#define CEC_OP_PROG_INFO_NONE_AVAILABLE                 0x0a
/* Not Programmed Error Info Operand (prog_error) */
#define CEC_OP_PROG_ERROR_NO_FREE_TIMER                 0x01
#define CEC_OP_PROG_ERROR_DATE_OUT_OF_RANGE             0x02
#define CEC_OP_PROG_ERROR_REC_SEQ_ERROR                 0x03
#define CEC_OP_PROG_ERROR_INV_EXT_PLUG                  0x04
#define CEC_OP_PROG_ERROR_INV_EXT_PHYS_ADDR             0x05
#define CEC_OP_PROG_ERROR_CA_UNSUPP                     0x06
#define CEC_OP_PROG_ERROR_INSUF_CA_ENTITLEMENTS         0x07
#define CEC_OP_PROG_ERROR_RESOLUTION_UNSUPP             0x08
#define CEC_OP_PROG_ERROR_PARENTAL_LOCK                 0x09
#define CEC_OP_PROG_ERROR_CLOCK_FAILURE                 0x0a
#define CEC_OP_PROG_ERROR_DUPLICATE                     0x0e

/* System Information Feature */
#define CEC_MSG_CEC_VERSION                             0x9e
/* CEC Version Operand (cec_version) */
#define CEC_OP_CEC_VERSION_1_3A                         4
#define CEC_OP_CEC_VERSION_1_4                          5
#define CEC_OP_CEC_VERSION_2_0                          6

#define CEC_MSG_GET_CEC_VERSION                         0x9f
#define CEC_MSG_GIVE_PHYSICAL_ADDR                      0x83
#define CEC_MSG_GET_MENU_LANGUAGE                       0x91
```

```
#define CEC_MSG_REPORT_PHYSICAL_ADDR                    0x84
/* Primary Device Type Operand (prim_devtype) */
#define CEC_OP_PRIM_DEVTYPE_TV                          0
#define CEC_OP_PRIM_DEVTYPE_RECORD                      1
#define CEC_OP_PRIM_DEVTYPE_TUNER                       3
#define CEC_OP_PRIM_DEVTYPE_PLAYBACK                    4
#define CEC_OP_PRIM_DEVTYPE_AUDIOSYSTEM                 5
#define CEC_OP_PRIM_DEVTYPE_SWITCH                      6
#define CEC_OP_PRIM_DEVTYPE_PROCESSOR                   7

#define CEC_MSG_SET_MENU_LANGUAGE                       0x32
#define CEC_MSG_REPORT_FEATURES                         0xa6    /* HDMI 2.0 */
/* All Device Types Operand (all_device_types) */
#define CEC_OP_ALL_DEVTYPE_TV                           0x80
#define CEC_OP_ALL_DEVTYPE_RECORD                       0x40
#define CEC_OP_ALL_DEVTYPE_TUNER                        0x20
#define CEC_OP_ALL_DEVTYPE_PLAYBACK                     0x10
#define CEC_OP_ALL_DEVTYPE_AUDIOSYSTEM                  0x08
#define CEC_OP_ALL_DEVTYPE_SWITCH                       0x04
/*
 * And if you wondering what happened to PROCESSOR devices: those should
 * be mapped to a SWITCH.
 */

/* Valid for RC Profile and Device Feature operands */
#define CEC_OP_FEAT_EXT                                 0x80    /* Extension bit */
/* RC Profile Operand (rc_profile) */
#define CEC_OP_FEAT_RC_TV_PROFILE_NONE                  0x00
#define CEC_OP_FEAT_RC_TV_PROFILE_1                     0x02
#define CEC_OP_FEAT_RC_TV_PROFILE_2                     0x06
#define CEC_OP_FEAT_RC_TV_PROFILE_3                     0x0a
#define CEC_OP_FEAT_RC_TV_PROFILE_4                     0x0e
#define CEC_OP_FEAT_RC_SRC_HAS_DEV_ROOT_MENU            0x50
#define CEC_OP_FEAT_RC_SRC_HAS_DEV_SETUP_MENU           0x48
#define CEC_OP_FEAT_RC_SRC_HAS_CONTENTS_MENU            0x44
#define CEC_OP_FEAT_RC_SRC_HAS_MEDIA_TOP_MENU           0x42
#define CEC_OP_FEAT_RC_SRC_HAS_MEDIA_CONTEXT_MENU       0x41
/* Device Feature Operand (dev_features) */
#define CEC_OP_FEAT_DEV_HAS_RECORD_TV_SCREEN            0x40
#define CEC_OP_FEAT_DEV_HAS_SET_OSD_STRING              0x20
#define CEC_OP_FEAT_DEV_HAS_DECK_CONTROL                0x10
#define CEC_OP_FEAT_DEV_HAS_SET_AUDIO_RATE              0x08
#define CEC_OP_FEAT_DEV_SINK_HAS_ARC_TX                 0x04
#define CEC_OP_FEAT_DEV_SOURCE_HAS_ARC_RX               0x02

#define CEC_MSG_GIVE_FEATURES                           0xa5    /* HDMI 2.0 */

/* Deck Control Feature */
#define CEC_MSG_DECK_CONTROL                            0x42
/* Deck Control Mode Operand (deck_control_mode) */
#define CEC_OP_DECK_CTL_MODE_SKIP_FWD                   1
#define CEC_OP_DECK_CTL_MODE_SKIP_REV                   2
#define CEC_OP_DECK_CTL_MODE_STOP                       3
#define CEC_OP_DECK_CTL_MODE_EJECT                      4

#define CEC_MSG_DECK_STATUS                             0x1b
/* Deck Info Operand (deck_info) */
#define CEC_OP_DECK_INFO_PLAY                           0x11
```

```
#define CEC_OP_DECK_INFO_RECORD                         0x12
#define CEC_OP_DECK_INFO_PLAY_REV                       0x13
#define CEC_OP_DECK_INFO_STILL                          0x14
#define CEC_OP_DECK_INFO_SLOW                           0x15
#define CEC_OP_DECK_INFO_SLOW_REV                       0x16
#define CEC_OP_DECK_INFO_FAST_FWD                       0x17
#define CEC_OP_DECK_INFO_FAST_REV                       0x18
#define CEC_OP_DECK_INFO_NO_MEDIA                       0x19
#define CEC_OP_DECK_INFO_STOP                           0x1a
#define CEC_OP_DECK_INFO_SKIP_FWD                       0x1b
#define CEC_OP_DECK_INFO_SKIP_REV                       0x1c
#define CEC_OP_DECK_INFO_INDEX_SEARCH_FWD               0x1d
#define CEC_OP_DECK_INFO_INDEX_SEARCH_REV               0x1e
#define CEC_OP_DECK_INFO_OTHER                          0x1f

#define CEC_MSG_GIVE_DECK_STATUS                        0x1a
/* Status Request Operand (status_req) */
#define CEC_OP_STATUS_REQ_ON                            1
#define CEC_OP_STATUS_REQ_OFF                           2
#define CEC_OP_STATUS_REQ_ONCE                          3

#define CEC_MSG_PLAY                                    0x41
/* Play Mode Operand (play_mode) */
#define CEC_OP_PLAY_MODE_PLAY_FWD                       0x24
#define CEC_OP_PLAY_MODE_PLAY_REV                       0x20
#define CEC_OP_PLAY_MODE_PLAY_STILL                     0x25
#define CEC_OP_PLAY_MODE_PLAY_FAST_FWD_MIN              0x05
#define CEC_OP_PLAY_MODE_PLAY_FAST_FWD_MED              0x06
#define CEC_OP_PLAY_MODE_PLAY_FAST_FWD_MAX              0x07
#define CEC_OP_PLAY_MODE_PLAY_FAST_REV_MIN              0x09
#define CEC_OP_PLAY_MODE_PLAY_FAST_REV_MED              0x0a
#define CEC_OP_PLAY_MODE_PLAY_FAST_REV_MAX              0x0b
#define CEC_OP_PLAY_MODE_PLAY_SLOW_FWD_MIN              0x15
#define CEC_OP_PLAY_MODE_PLAY_SLOW_FWD_MED              0x16
#define CEC_OP_PLAY_MODE_PLAY_SLOW_FWD_MAX              0x17
#define CEC_OP_PLAY_MODE_PLAY_SLOW_REV_MIN              0x19
#define CEC_OP_PLAY_MODE_PLAY_SLOW_REV_MED              0x1a
#define CEC_OP_PLAY_MODE_PLAY_SLOW_REV_MAX              0x1b

/* Tuner Control Feature */
#define CEC_MSG_GIVE_TUNER_DEVICE_STATUS                0x08
#define CEC_MSG_SELECT_ANALOGUE_SERVICE                 0x92
#define CEC_MSG_SELECT_DIGITAL_SERVICE                  0x93
#define CEC_MSG_TUNER_DEVICE_STATUS                     0x07
/* Recording Flag Operand (rec_flag) */
#define CEC_OP_REC_FLAG_USED                            0
#define CEC_OP_REC_FLAG_NOT_USED                        1
/* Tuner Display Info Operand (tuner_display_info) */
#define CEC_OP_TUNER_DISPLAY_INFO_DIGITAL               0
#define CEC_OP_TUNER_DISPLAY_INFO_NONE                  1
#define CEC_OP_TUNER_DISPLAY_INFO_ANALOGUE              2

#define CEC_MSG_TUNER_STEP_DECREMENT                    0x06
#define CEC_MSG_TUNER_STEP_INCREMENT                    0x05

/* Vendor Specific Commands Feature */

/*
```

```
 * Has also:
 *      CEC_MSG_CEC_VERSION
 *      CEC_MSG_GET_CEC_VERSION
 */
#define CEC_MSG_DEVICE_VENDOR_ID                        0x87
#define CEC_MSG_GIVE_DEVICE_VENDOR_ID                   0x8c
#define CEC_MSG_VENDOR_COMMAND                          0x89
#define CEC_MSG_VENDOR_COMMAND_WITH_ID                  0xa0
#define CEC_MSG_VENDOR_REMOTE_BUTTON_DOWN               0x8a
#define CEC_MSG_VENDOR_REMOTE_BUTTON_UP                 0x8b

/* OSD Display Feature */
#define CEC_MSG_SET_OSD_STRING                          0x64
/* Display Control Operand (disp_ctl) */
#define CEC_OP_DISP_CTL_DEFAULT                         0x00
#define CEC_OP_DISP_CTL_UNTIL_CLEARED                   0x40
#define CEC_OP_DISP_CTL_CLEAR                           0x80

/* Device OSD Transfer Feature */
#define CEC_MSG_GIVE_OSD_NAME                           0x46
#define CEC_MSG_SET_OSD_NAME                            0x47

/* Device Menu Control Feature */
#define CEC_MSG_MENU_REQUEST                            0x8d
/* Menu Request Type Operand (menu_req) */
#define CEC_OP_MENU_REQUEST_ACTIVATE                    0x00
#define CEC_OP_MENU_REQUEST_DEACTIVATE                  0x01
#define CEC_OP_MENU_REQUEST_QUERY                       0x02

#define CEC_MSG_MENU_STATUS                             0x8e
/* Menu State Operand (menu_state) */
#define CEC_OP_MENU_STATE_ACTIVATED                     0x00
#define CEC_OP_MENU_STATE_DEACTIVATED                   0x01

#define CEC_MSG_USER_CONTROL_PRESSED                    0x44
/* UI Broadcast Type Operand (ui_bcast_type) */
#define CEC_OP_UI_BCAST_TYPE_TOGGLE_ALL                 0x00
#define CEC_OP_UI_BCAST_TYPE_TOGGLE_DIG_ANA             0x01
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE                   0x10
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE_T                 0x20
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE_CABLE             0x30
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE_SAT               0x40
#define CEC_OP_UI_BCAST_TYPE_DIGITAL                    0x50
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_T                  0x60
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_CABLE              0x70
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_SAT                0x80
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_COM_SAT            0x90
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_COM_SAT2           0x91
#define CEC_OP_UI_BCAST_TYPE_IP                         0xa0
/* UI Sound Presentation Control Operand (ui_snd_pres_ctl) */
#define CEC_OP_UI_SND_PRES_CTL_DUAL_MONO                0x10
#define CEC_OP_UI_SND_PRES_CTL_KARAOKE                  0x20
#define CEC_OP_UI_SND_PRES_CTL_DOWNMIX                  0x80
#define CEC_OP_UI_SND_PRES_CTL_REVERB                   0x90
#define CEC_OP_UI_SND_PRES_CTL_EQUALIZER                0xa0
#define CEC_OP_UI_SND_PRES_CTL_BASS_UP                  0xb1
#define CEC_OP_UI_SND_PRES_CTL_BASS_NEUTRAL             0xb2
#define CEC_OP_UI_SND_PRES_CTL_BASS_DOWN                0xb3
```

```
#define CEC_OP_UI_SND_PRES_CTL_TREBLE_UP                0xc1
#define CEC_OP_UI_SND_PRES_CTL_TREBLE_NEUTRAL           0xc2
#define CEC_OP_UI_SND_PRES_CTL_TREBLE_DOWN              0xc3

#define CEC_MSG_USER_CONTROL_RELEASED                   0x45

/* Remote Control Passthrough Feature */

/*
 * Has also:
 *      CEC_MSG_USER_CONTROL_PRESSED
 *      CEC_MSG_USER_CONTROL_RELEASED
 */

/* Power Status Feature */
#define CEC_MSG_GIVE_DEVICE_POWER_STATUS                0x8f
#define CEC_MSG_REPORT_POWER_STATUS                     0x90
/* Power Status Operand (pwr_state) */
#define CEC_OP_POWER_STATUS_ON                          0
#define CEC_OP_POWER_STATUS_STANDBY                     1
#define CEC_OP_POWER_STATUS_TO_ON                       2
#define CEC_OP_POWER_STATUS_TO_STANDBY                  3

/* General Protocol Messages */
#define CEC_MSG_FEATURE_ABORT                           0x00
/* Abort Reason Operand (reason) */
#define CEC_OP_ABORT_UNRECOGNIZED_OP                    0
#define CEC_OP_ABORT_INCORRECT_MODE                     1
#define CEC_OP_ABORT_NO_SOURCE                          2
#define CEC_OP_ABORT_INVALID_OP                         3
#define CEC_OP_ABORT_REFUSED                            4
#define CEC_OP_ABORT_UNDETERMINED                       5

#define CEC_MSG_ABORT                                   0xff

/* System Audio Control Feature */

/*
 * Has also:
 *      CEC_MSG_USER_CONTROL_PRESSED
 *      CEC_MSG_USER_CONTROL_RELEASED
 */
#define CEC_MSG_GIVE_AUDIO_STATUS                       0x71
#define CEC_MSG_GIVE_SYSTEM_AUDIO_MODE_STATUS           0x7d
#define CEC_MSG_REPORT_AUDIO_STATUS                     0x7a
/* Audio Mute Status Operand (aud_mute_status) */
#define CEC_OP_AUD_MUTE_STATUS_OFF                      0
#define CEC_OP_AUD_MUTE_STATUS_ON                       1

#define CEC_MSG_REPORT_SHORT_AUDIO_DESCRIPTOR           0xa3
#define CEC_MSG_REQUEST_SHORT_AUDIO_DESCRIPTOR          0xa4
#define CEC_MSG_SET_SYSTEM_AUDIO_MODE                   0x72
/* System Audio Status Operand (sys_aud_status) */
#define CEC_OP_SYS_AUD_STATUS_OFF                       0
#define CEC_OP_SYS_AUD_STATUS_ON                        1

#define CEC_MSG_SYSTEM_AUDIO_MODE_REQUEST               0x70
#define CEC_MSG_SYSTEM_AUDIO_MODE_STATUS                0x7e
```

```
/* Audio Format ID Operand (audio_format_id) */
#define CEC_OP_AUD_FMT_ID_CEA861                        0
#define CEC_OP_AUD_FMT_ID_CEA861_CXT                    1

/* Audio Rate Control Feature */
#define CEC_MSG_SET_AUDIO_RATE                          0x9a
/* Audio Rate Operand (audio_rate) */
#define CEC_OP_AUD_RATE_OFF                             0
#define CEC_OP_AUD_RATE_WIDE_STD                        1
#define CEC_OP_AUD_RATE_WIDE_FAST                       2
#define CEC_OP_AUD_RATE_WIDE_SLOW                       3
#define CEC_OP_AUD_RATE_NARROW_STD                      4
#define CEC_OP_AUD_RATE_NARROW_FAST                     5
#define CEC_OP_AUD_RATE_NARROW_SLOW                     6

/* Audio Return Channel Control Feature */
#define CEC_MSG_INITIATE_ARC                            0xc0
#define CEC_MSG_REPORT_ARC_INITIATED                    0xc1
#define CEC_MSG_REPORT_ARC_TERMINATED                   0xc2
#define CEC_MSG_REQUEST_ARC_INITIATION                  0xc3
#define CEC_MSG_REQUEST_ARC_TERMINATION                 0xc4
#define CEC_MSG_TERMINATE_ARC                           0xc5

/* Dynamic Audio Lipsync Feature */
/* Only for CEC 2.0 and up */
#define CEC_MSG_REQUEST_CURRENT_LATENCY                 0xa7
#define CEC_MSG_REPORT_CURRENT_LATENCY                  0xa8
/* Low Latency Mode Operand (low_latency_mode) */
#define CEC_OP_LOW_LATENCY_MODE_OFF                     0
#define CEC_OP_LOW_LATENCY_MODE_ON                      1
/* Audio Output Compensated Operand (audio_out_compensated) */
#define CEC_OP_AUD_OUT_COMPENSATED_NA                   0
#define CEC_OP_AUD_OUT_COMPENSATED_DELAY                1
#define CEC_OP_AUD_OUT_COMPENSATED_NO_DELAY             2
#define CEC_OP_AUD_OUT_COMPENSATED_PARTIAL_DELAY        3

/* Capability Discovery and Control Feature */
#define CEC_MSG_CDC_MESSAGE                             0xf8
/* Ethernet-over-HDMI: nobody ever does this... */
#define CEC_MSG_CDC_HEC_INQUIRE_STATE                   0x00
#define CEC_MSG_CDC_HEC_REPORT_STATE                    0x01
/* HEC Functionality State Operand (hec_func_state) */
#define CEC_OP_HEC_FUNC_STATE_NOT_SUPPORTED             0
#define CEC_OP_HEC_FUNC_STATE_INACTIVE                  1
#define CEC_OP_HEC_FUNC_STATE_ACTIVE                    2
#define CEC_OP_HEC_FUNC_STATE_ACTIVATION_FIELD          3
/* Host Functionality State Operand (host_func_state) */
#define CEC_OP_HOST_FUNC_STATE_NOT_SUPPORTED            0
#define CEC_OP_HOST_FUNC_STATE_INACTIVE                 1
#define CEC_OP_HOST_FUNC_STATE_ACTIVE                   2
/* ENC Functionality State Operand (enc_func_state) */
#define CEC_OP_ENC_FUNC_STATE_EXT_CON_NOT_SUPPORTED     0
#define CEC_OP_ENC_FUNC_STATE_EXT_CON_INACTIVE          1
#define CEC_OP_ENC_FUNC_STATE_EXT_CON_ACTIVE            2
/* CDC Error Code Operand (cdc_errcode) */
#define CEC_OP_CDC_ERROR_CODE_NONE                      0
#define CEC_OP_CDC_ERROR_CODE_CAP_UNSUPPORTED           1
#define CEC_OP_CDC_ERROR_CODE_WRONG_STATE               2
```

```
#define CEC_OP_CDC_ERROR_CODE_OTHER                     3
/* HEC Support Operand (hec_support) */
#define CEC_OP_HEC_SUPPORT_NO                           0
#define CEC_OP_HEC_SUPPORT_YES                          1
/* HEC Activation Operand (hec_activation) */
#define CEC_OP_HEC_ACTIVATION_ON                        0
#define CEC_OP_HEC_ACTIVATION_OFF                       1

#define CEC_MSG_CDC_HEC_SET_STATE_ADJACENT              0x02
#define CEC_MSG_CDC_HEC_SET_STATE                       0x03
/* HEC Set State Operand (hec_set_state) */
#define CEC_OP_HEC_SET_STATE_DEACTIVATE                 0
#define CEC_OP_HEC_SET_STATE_ACTIVATE                   1

#define CEC_MSG_CDC_HEC_REQUEST_DEACTIVATION            0x04
#define CEC_MSG_CDC_HEC_NOTIFY_ALIVE                    0x05
#define CEC_MSG_CDC_HEC_DISCOVER                        0x06
/* Hotplug Detect messages */
#define CEC_MSG_CDC_HPD_SET_STATE                       0x10
/* HPD State Operand (hpd_state) */
#define CEC_OP_HPD_STATE_CP_EDID_DISABLE                0
#define CEC_OP_HPD_STATE_CP_EDID_ENABLE                 1
#define CEC_OP_HPD_STATE_CP_EDID_DISABLE_ENABLE         2
#define CEC_OP_HPD_STATE_EDID_DISABLE                   3
#define CEC_OP_HPD_STATE_EDID_ENABLE                    4
#define CEC_OP_HPD_STATE_EDID_DISABLE_ENABLE            5
#define CEC_MSG_CDC_HPD_REPORT_STATE                    0x11
/* HPD Error Code Operand (hpd_error) */
#define CEC_OP_HPD_ERROR_NONE                           0
#define CEC_OP_HPD_ERROR_INITIATOR_NOT_CAPABLE          1
#define CEC_OP_HPD_ERROR_INITIATOR_WRONG_STATE          2
#define CEC_OP_HPD_ERROR_OTHER                          3
#define CEC_OP_HPD_ERROR_NONE_NO_VIDEO                  4

/* End of Messages */

/* Helper functions to identify the `special' CEC devices */

static inline int cec_is_2nd_tv(const struct  cec_log_addrs
 *las)
{
        /*
         * It is a second TV if the logical address is 14 or 15 and the
         * primary device type is a TV.
         */
        return las->num_log_addrs &&
               las->log_addr[0] >= CEC_LOG_ADDR_SPECIFIC &&
               las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_TV;
}

static inline int cec_is_processor(const struct  cec_log_addrs
 *las)
{
        /*
         * It is a processor if the logical address is 12-15 and the
         * primary device type is a Processor.
         */
        return las->num_log_addrs &&
```

```
               las->log_addr[0] >= CEC_LOG_ADDR_BACKUP_1 &&
               las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_PROCESSOR;
}

static inline int cec_is_switch(const struct  cec_log_addrs
 *las)
{
        /*
         * It is a switch if the logical address is 15 and the
         * primary device type is a Switch and the CDC-Only flag is not set.
         */
        return las->num_log_addrs == 1 &&
               las->log_addr[0] == CEC_LOG_ADDR_UNREGISTERED &&
               las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_SWITCH &&
               !(las->flags &  CEC_LOG_ADDRS_FL_CDC_ONLY );
}

static inline int cec_is_cdc_only(const struct  cec_log_addrs
 *las)
{
        /*
         * It is a CDC-only device if the logical address is 15 and the
         * primary device type is a Switch and the CDC-Only flag is set.
         */
        return las->num_log_addrs == 1 &&
               las->log_addr[0] == CEC_LOG_ADDR_UNREGISTERED &&
               las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_SWITCH &&
               (las->flags &  CEC_LOG_ADDRS_FL_CDC_ONLY );
}

#endif
```

1.6.4 Revision and Copyright

Authors:

- Verkuil, Hans <hans.verkuil@cisco.com>
- Initial version.

1.6.5 Revision History

revision 1.0.0 / 2016-03-17 (*hv*)

Initial revision

1.7 Generic Error Codes

Table 1.235: Generic error codes

EAGAIN (aka EWOULDBLOCK)	The ioctl can't be handled because the device is in state where it can't perform it. This could happen for example in case where device is sleeping and ioctl is performed to query statistics. It is also returned when the ioctl would need to wait for an event, but the device was opened in non-blocking mode.
EBADF	The file descriptor is not a valid.
EBUSY	The ioctl can't be handled because the device is busy. This is typically return while device is streaming, and an ioctl tried to change something that would affect the stream, or would require the usage of a hardware resource that was already allocated. The ioctl must not be retried without performing another action to fix the problem first (typically: stop the stream before retrying).
EFAULT	There was a failure while copying data from/to userspace, probably caused by an invalid pointer reference.
EINVAL	One or more of the ioctl parameters are invalid or out of the allowed range. This is a widely used error code. See the individual ioctl requests for specific causes.
ENODEV	Device not found or was removed.
ENOMEM	There's not enough memory to handle the desired operation.
ENOTTY	The ioctl is not supported by the driver, actually meaning that the required functionality is not available, or the file descriptor is not for a media device.
ENOSPC	On USB devices, the stream ioctl's can return this error, meaning that this request would overcommit the usb bandwidth reserved for periodic transfers (up to 80% of the USB bandwidth).
EPERM	Permission denied. Can be returned if the device needs write permission, or some special capabilities is needed (e. g. root)
EIO	I/O error. Typically used when there are problems communicating with a hardware device. This could indicate broken or flaky hardware. It's a 'Something is wrong, I give up!' type of error.

Note:

1. *This list is not exhaustive; ioctls may return other error codes. Since errors may have side effects such as a driver reset, applications should abort on unexpected errors, or otherwise assume that the device is in a bad state.*
2. *Request-specific error codes are listed in the individual requests descriptions.*

1.8 GNU Free Documentation License

1.8.1 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work,

regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1.8.2 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language. A "Secondary Section" is a named appendix or a front-matter section of the *Document* that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them. The "Invariant Sections" are certain *Secondary Sections* whose titles are designated, as being those of Invariant Sections, in the notice that says that the *Document* is released under this License. The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the *Document* is released under this License. A "Transparent" copy of the *Document* means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only. The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

1.8.3 2. VERBATIM COPYING

You may copy and distribute the *Document* in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in *section 3* .

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

1.8.4 3. COPYING IN QUANTITY

If you publish printed copies of the *Document* numbering more than 100, and the Document's license notice requires *Cover Texts* , you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material

on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the *Document* and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute *Opaque* copies of the *Document* numbering more than 100, you must either include a machine-readable *Transparent* copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the *Document* well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

1.8.5 4. MODIFICATIONS

You may copy and distribute a *Modified Version* of the *Document* under the conditions of sections *2* and *3* above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the *Title Page* (and on the covers, if any) a title distinct from that of the *Document* , and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the *Title Page* , as authors, one or more persons or entities responsible for authorship of the modifications in the *Modified Version* , together with at least five of the principal authors of the *Document* (all of its principal authors, if it has less than five).
- **C.** State on the *Title Page* the name of the publisher of the *Modified Version* , as the publisher.
- **D.** Preserve all the copyright notices of the *Document* .
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the *Modified Version* under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of *Invariant Sections* and required *Cover Texts* given in the *Document's* license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the *Modified Version* as given on the *Title Page* . If there is no section entitled "History" in the *Document* , create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the *Document* for public access to a *Transparent* copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements

and/or dedications given therein.

- **L.** Preserve all the *Invariant Sections* of the *Document* , unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section entitled “Endorsements”. Such a section may not be included in the *Modified Version* .
- **N.** Do not retitle any existing section as “Endorsements” or to conflict in title with any *Invariant Section* .

If the *Modified Version* includes new front-matter sections or appendices that qualify as *Secondary Sections* and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of *Invariant Sections* in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your *Modified Version* by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a *Front-Cover Text* , and a passage of up to 25 words as a *Back-Cover Text* , to the end of the list of *Cover Texts* in the *Modified Version* . Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the *Document* already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the *Document* do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any *Modified Version* .

1.8.6 5. COMBINING DOCUMENTS

You may combine the *Document* with other documents released under this License, under the terms defined in *section 4* above for modified versions, provided that you include in the combination all of the *Invariant Sections* of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical *Invariant Sections* may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

1.8.7 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the *Document* and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and dispbibute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

1.8.8 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the *Document* or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a *Modified Version* of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document , on account of their being thus compiled, if they are not themselves derivative works of the Document. If the *Cover Text* requirement of *section 3* is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

1.8.9 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the *Document* under the terms of *section 4* . Replacing *Invariant Sections* with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

1.8.10 9. TERMINATION

You may not copy, modify, sublicense, or distribute the *Document* except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

1.8.11 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the *Document* specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

1.8.12 Addendum

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME.
>
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the *Invariant Sections* being LIST THEIR TITLES, with the *Front-Cover Texts* being LIST, and with the *Back-Cover Texts* being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no *Invariant Sections* , write “with no Invariant Sections” instead of saying which ones are invariant. If you have no *Front-Cover Texts* , write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST”; likewise for *Back-Cover Texts* .

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

CHAPTER

TWO

MEDIA SUBSYSTEM KERNEL INTERNAL API

Table of Contents

2.1 Video2Linux devices

2.1.1 Introduction

The V4L2 drivers tend to be very complex due to the complexity of the hardware: most devices have multiple ICs, export multiple device nodes in /dev, and create also non-V4L2 devices such as DVB, ALSA, FB, I2C and input (IR) devices.

Especially the fact that V4L2 drivers have to setup supporting ICs to do audio/video muxing/encoding/decoding makes it more complex than most. Usually these ICs are connected to the main bridge driver through one or more I2C busses, but other busses can also be used. Such devices are called 'sub-devices'.

For a long time the framework was limited to the video_device struct for creating V4L device nodes and video_buf for handling the video buffers (note that this document does not discuss the video_buf framework).

This meant that all drivers had to do the setup of device instances and connecting to sub-devices themselves. Some of this is quite complicated to do right and many drivers never did do it correctly.

There is also a lot of common code that could never be refactored due to the lack of a framework.

So this framework sets up the basic building blocks that all drivers need and this same framework should make it much easier to refactor common code into utility functions shared by all drivers.

A good example to look at as a reference is the v4l2-pci-skeleton.c source that is available in samples/v4l/. It is a skeleton driver for a PCI capture card, and demonstrates how to use the V4L2 driver framework. It can be used as a template for real PCI video capture driver.

2.1.2 Structure of a V4L driver

All drivers have the following structure:

1. A struct for each device instance containing the device state.

2. A way of initializing and commanding sub-devices (if any).
3. Creating V4L2 device nodes (/dev/videoX, /dev/vbiX and /dev/radioX) and keeping track of device-node specific data.
4. Filehandle-specific structs containing per-filehandle data;
5. video buffer handling.

This is a rough schematic of how it all relates:

```
device instances
  |
  +-sub-device instances
  |
  \-V4L2 device nodes
      |
      \-filehandle instances
```

2.1.3 Structure of the V4L2 framework

The framework closely resembles the driver structure: it has a v4l2_device struct for the device instance data, a v4l2_subdev struct to refer to sub-device instances, the video_device struct stores V4L2 device node data and the v4l2_fh struct keeps track of filehandle instances.

The V4L2 framework also optionally integrates with the media framework. If a driver sets the struct v4l2_device mdev field, sub-devices and video nodes will automatically appear in the media framework as entities.

2.1.4 Video device' s internal representation

The actual device nodes in the `/dev` directory are created using the *video_device* struct (`v4l2-dev.h`). This struct can either be allocated dynamically or embedded in a larger struct.

To allocate it dynamically use *video_device_alloc()*:

```
struct video_device *vdev = video_device_alloc();

if (vdev == NULL)
        return -ENOMEM;

vdev->release = video_device_release;
```

If you embed it in a larger struct, then you must set the `release()` callback to your own function:

```
struct video_device *vdev = &my_vdev->vdev;

vdev->release = my_vdev_release;
```

The `release()` callback must be set and it is called when the last user of the video device exits.

The default *video_device_release()* callback currently just calls `kfree` to free the allocated memory.

There is also a :*video_device_release_empty()* function that does nothing (is empty) and should be used if the struct is embedded and there is nothing to do when it is released.

You should also set these fields of *video_device*:

- *video_device*->v4l2_dev: must be set to the *v4l2_device* parent device.
- *video_device*->name: set to something descriptive and unique.

- *video_device*->vfl_dir: set this to VFL_DIR_RX for capture devices (VFL_DIR_RX has value 0, so this is normally already the default), set to VFL_DIR_TX for output devices and VFL_DIR_M2M for mem2mem (codec) devices.
- *video_device*->fops: set to the *v4l2_file_operations* struct.
- *video_device*->ioctl_ops: if you use the *v4l2_ioctl_ops* to simplify ioctl maintenance (highly recommended to use this and it might become compulsory in the future!), then set this to your *v4l2_ioctl_ops* struct. The *video_device*->vfl_type and *video_device*->vfl_dir fields are used to disable ops that do not match the type/dir combination. E.g. VBI ops are disabled for non-VBI nodes, and output ops are disabled for a capture device. This makes it possible to provide just one *v4l2_ioctl_ops* struct for both vbi and video nodes.
- *video_device*->lock: leave to NULL if you want to do all the locking in the driver. Otherwise you give it a pointer to a struct mutex_lock and before the *video_device*->unlocked_ioctl file operation is called this lock will be taken by the core and released afterwards. See the next section for more details.
- *video_device*->queue: a pointer to the struct *vb2_queue* associated with this device node. If queue is not NULL, and queue->lock is not NULL, then queue->lock is used for the queuing ioctls (VIDIOC_REQBUFS, CREATE_BUFS, QBUF, DQBUF, QUERYBUF, PREPARE_BUF, STREAMON and STREAMOFF) instead of the lock above. That way the *vb2* queuing framework does not have to wait for other ioctls. This queue pointer is also used by the *vb2* helper functions to check for queuing ownership (i.e. is the filehandle calling it allowed to do the operation).
- *video_device*->prio: keeps track of the priorities. Used to implement VIDIOC_G_PRIORITY and VIDIOC_S_PRIORITY. If left to NULL, then it will use the struct *v4l2_prio_state* in *v4l2_device*. If you want to have a separate priority state per (group of) device node(s), then you can point it to your own struct *v4l2_prio_state*.
- *video_device*->dev_parent: you only set this if v4l2_device was registered with NULL as the parent device struct. This only happens in cases where one hardware device has multiple PCI devices that all share the same *v4l2_device* core.

  The cx88 driver is an example of this: one core *v4l2_device* struct, but it is used by both a raw video PCI device (cx8800) and a MPEG PCI device (cx8802). Since the *v4l2_device* cannot be associated with two PCI devices at the same time it is setup without a parent device. But when the struct *video_device* is initialized you **do** know which parent PCI device to use and so you set dev_device to the correct PCI device.

If you use *v4l2_ioctl_ops*, then you should set *video_device*->unlocked_ioctl to *video_ioctl2()* in your *v4l2_file_operations* struct.

In some cases you want to tell the core that a function you had specified in your *v4l2_ioctl_ops* should be ignored. You can mark such ioctls by calling this function before *video_register_device()* is called:

v4l2_disable_ioctl (*vdev*, cmd).

This tends to be needed if based on external factors (e.g. which card is being used) you want to turns off certain features in *v4l2_ioctl_ops* without having to make a new struct.

The *v4l2_file_operations* struct is a subset of file_operations. The main difference is that the inode argument is omitted since it is never used.

If integration with the media framework is needed, you must initialize the *media_entity* struct embedded in the *video_device* struct (entity field) by calling *media_entity_pads_init()*:

```
struct media_pad *pad = &my_vdev->pad;
int err;

err = media_entity_pads_init(&vdev->entity, 1, pad);
```

The pads array must have been previously initialized. There is no need to manually set the struct media_entity type and name fields.

A reference to the entity will be automatically acquired/released when the video device is opened/closed.

ioctls and locking

The V4L core provides optional locking services. The main service is the lock field in struct *video_device*, which is a pointer to a mutex. If you set this pointer, then that will be used by unlocked_ioctl to serialize all ioctls.

If you are using the *videobuf2 framework* , then there is a second lock that you can set: *video_device*->queue->lock. If set, then this lock will be used instead of *video_device*->lock to serialize all queuing ioctls (see the previous section for the full list of those ioctls).

The advantage of using a different lock for the queuing ioctls is that for some drivers (particularly USB drivers) certain commands such as setting controls can take a long time, so you want to use a separate lock for the buffer queuing ioctls. That way your VIDIOC_DQBUF doesn't stall because the driver is busy changing the e.g. exposure of the webcam.

Of course, you can always do all the locking yourself by leaving both lock pointers at NULL.

If you use the old *videobuf framework* then you must pass the *video_device*->lock to the videobuf queue initialize function: if videobuf has to wait for a frame to arrive, then it will temporarily unlock the lock and relock it afterwards. If your driver also waits in the code, then you should do the same to allow other processes to access the device node while the first process is waiting for something.

In the case of *videobuf2* you will need to implement the wait_prepare() and wait_finish() callbacks to unlock/lock if applicable. If you use the queue->lock pointer, then you can use the helper functions *vb2_ops_wait_prepare()* and *vb2_ops_wait_finish()*.

The implementation of a hotplug disconnect should also take the lock from *video_device* before calling v4l2_device_disconnect. If you are also using *video_device*->queue->lock, then you have to first lock *video_device*->queue->lock followed by *video_device*->lock. That way you can be sure no ioctl is running when you call *v4l2_device_disconnect()*.

Video device registration

Next you register the video device with *video_register_device()*. This will create the character device for you.

```
err = video_register_device(vdev, VFL_TYPE_GRABBER, -1);
if (err) {
        video_device_release(vdev); /* or kfree(my_vdev); */
        return err;
}
```

If the *v4l2_device* parent device has a not NULL mdev field, the video device entity will be automatically registered with the media device.

Which device is registered depends on the type argument. The following types exist:

- VFL_TYPE_GRABBER: /dev/videoX for video input/output devices
- VFL_TYPE_VBI: /dev/vbiX for vertical blank data (i.e. closed captions, teletext)
- VFL_TYPE_RADIO: /dev/radioX for radio tuners
- VFL_TYPE_SDR: /dev/swradioX for Software Defined Radio tuners
- VFL_TYPE_TOUCH: /dev/v4l-touchX for touch sensors

The last argument gives you a certain amount of control over the device device node number used (i.e. the X in videoX). Normally you will pass -1 to let the v4l2 framework pick the first free number. But sometimes users want to select a specific node number. It is common that drivers allow the user to select a specific device node number through a driver module option. That number is then passed to this function and video_register_device will attempt to select that device node number. If that number was already in use, then the next free device node number will be selected and it will send a warning to the kernel log.

Another use-case is if a driver creates many devices. In that case it can be useful to place different video devices in separate ranges. For example, video capture devices start at 0, video output devices start at 16. So you can use the last argument to specify a minimum device node number and the v4l2 framework will try to pick the first free number that is equal or higher to what you passed. If that fails, then it will just pick the first free number.

Since in this case you do not care about a warning about not being able to select the specified device node number, you can call the function *video_register_device_no_warn()* instead.

Whenever a device node is created some attributes are also created for you. If you look in /sys/class/video4linux you see the devices. Go into e.g. video0 and you will see 'name', 'dev_debug' and 'index' attributes. The 'name' attribute is the 'name' field of the video_device struct. The 'dev_debug' attribute can be used to enable core debugging. See the next section for more detailed information on this.

The 'index' attribute is the index of the device node: for each call to *video_register_device()* the index is just increased by 1. The first video device node you register always starts with index 0.

Users can setup udev rules that utilize the index attribute to make fancy device names (e.g. 'mpegX' for MPEG video capture device nodes).

After the device was successfully registered, then you can use these fields:

- *video_device*->vfl_type: the device type passed to *video_register_device()*.
- *video_device*->minor: the assigned device minor number.
- *video_device*->num: the device node number (i.e. the X in videoX).
- *video_device*->index: the device index number.

If the registration failed, then you need to call *video_device_release()* to free the allocated *video_device* struct, or free your own struct if the *video_device* was embedded in it. The vdev->release() callback will never be called if the registration failed, nor should you ever attempt to unregister the device if the registration failed.

video device debugging

The 'dev_debug' attribute that is created for each video, vbi, radio or swradio device in /sys/class/video4linux/<devX>/ allows you to enable logging of file operations.

It is a bitmask and the following bits can be set:

Mask	Description
0x01	Log the ioctl name and error code. VIDIOC_(D)QBUF ioctls are only logged if bit 0x08 is also set.
0x02	Log the ioctl name arguments and error code. VIDIOC_(D)QBUF ioctls are only logged if bit 0x08 is also set.
0x04	Log the file operations open, release, read, write, mmap and get_unmapped_area. The read and write operations are only logged if bit 0x08 is also set.
0x08	Log the read and write file operations and the VIDIOC_QBUF and VIDIOC_DQBUF ioctls.
0x10	Log the poll file operation.

Video device cleanup

When the video device nodes have to be removed, either during the unload of the driver or because the USB device was disconnected, then you should unregister them with:

video_unregister_device() (*vdev*);

This will remove the device nodes from sysfs (causing udev to remove them from /dev).

After *video_unregister_device()* returns no new opens can be done. However, in the case of USB devices some application might still have one of these device nodes open. So after the unregister all file operations (except release, of course) will return an error as well.

When the last user of the video device node exits, then the `vdev->release()` callback is called and you can do the final cleanup there.

Don't forget to cleanup the media entity associated with the video device if it has been initialized:

media_entity_cleanup (&vdev->entity);

This can be done from the release callback.

helper functions

There are a few useful helper functions:

- file and *video_device* private data

You can set/get driver private data in the video_device struct using:

video_get_drvdata (*vdev*);

video_set_drvdata (*vdev*);

Note that you can safely call *video_set_drvdata()* before calling *video_register_device()*.

And this function:

video_devdata (struct file *file);

returns the video_device belonging to the file struct.

The *video_devdata()* function combines *video_get_drvdata()* with *video_devdata()*:

video_drvdata (struct file *file);

You can go from a *video_device* struct to the v4l2_device struct using:

```
struct v4l2_device *v4l2_dev = vdev->v4l2_dev;
```

- Device node name

The *video_device* node kernel name can be retrieved using:

video_device_node_name (*vdev*);

The name is used as a hint by userspace tools such as udev. The function should be used where possible instead of accessing the video_device::num and video_device::minor fields.

video_device functions and data structures

struct **v4l2_prio_state**
stores the priority states

Definition

```
struct v4l2_prio_state {
  atomic_t prios[4];
};
```

Members

prios[4] array with elements to store the array priorities

Description

Note:

*The size of **prios** array matches the number of priority types defined by enum* `v4l2_priority`.

void **v4l2_prio_init**(struct *v4l2_prio_state* * *global*)
initializes a struct v4l2_prio_state

Parameters

struct v4l2_prio_state * global pointer to *struct v4l2_prio_state*

int **v4l2_prio_change**(struct *v4l2_prio_state* * *global*, enum *v4l2_priority* * *local*, enum *v4l2_priority new*)
changes the v4l2 file handler priority

Parameters

struct v4l2_prio_state * global pointer to the *struct v4l2_prio_state* of the device node.

enum v4l2_priority * local pointer to the desired priority, as defined by enum *v4l2_priority*

enum v4l2_priority new Priority type requested, as defined by enum *v4l2_priority*.

Description

Note:

This function should be used only by the V4L2 core.

void **v4l2_prio_open**(struct *v4l2_prio_state* * *global*, enum *v4l2_priority* * *local*)
Implements the priority logic for a file handler open

Parameters

struct v4l2_prio_state * global pointer to the *struct v4l2_prio_state* of the device node.

enum v4l2_priority * local pointer to the desired priority, as defined by enum *v4l2_priority*

Description

Note:

This function should be used only by the V4L2 core.

void **v4l2_prio_close**(struct *v4l2_prio_state* * *global*, enum *v4l2_priority local*)
Implements the priority logic for a file handler close

Parameters

struct v4l2_prio_state * global pointer to the *struct v4l2_prio_state* of the device node.

enum v4l2_priority local priority to be released, as defined by enum *v4l2_priority*

Description

Note:

This function should be used only by the V4L2 core.

enum *v4l2_priority* **v4l2_prio_max**(struct *v4l2_prio_state* * *global*)
Return the maximum priority, as stored at the **global** array.

Parameters

struct v4l2_prio_state * global pointer to the *struct v4l2_prio_state* of the device node.

Description

Note:

This function should be used only by the V4L2 core.

int **v4l2_prio_check**(struct *v4l2_prio_state * global*, enum *v4l2_priority local*)
Implements the priority logic for a file handler close

Parameters

struct v4l2_prio_state * global pointer to the *struct v4l2_prio_state* of the device node.

enum v4l2_priority local desired priority, as defined by enum *v4l2_priority* local

Description

Note:

This function should be used only by the V4L2 core.

struct **v4l2_file_operations**
fs operations used by a V4L2 device

Definition

```
struct v4l2_file_operations {
  struct module * owner;
  ssize_t (* read) (struct file *, char __user *, size_t, loff_t *);
  ssize_t (* write) (struct file *, const char __user *, size_t, loff_t *);
  unsigned int (* poll) (struct file *, struct poll_table_struct *);
  long (* unlocked_ioctl) (struct file *, unsigned int, unsigned long);
#ifdef CONFIG_COMPAT
  long (* compat_ioctl32) (struct file *, unsigned int, unsigned long);
#endif
  unsigned long (* get_unmapped_area) (struct file *, unsigned long,unsigned long, unsigned␣
 ↪long, unsigned long);
  int (* mmap) (struct file *, struct vm_area_struct *);
  int (* open) (struct file *);
  int (* release) (struct file *);
};
```

Members

owner pointer to struct module

read operations needed to implement the `read()` syscall

write operations needed to implement the *write()* syscall

poll operations needed to implement the `poll()` syscall

unlocked_ioctl operations needed to implement the `ioctl()` syscall

compat_ioctl32 operations needed to implement the `ioctl()` syscall for the special case where the Kernel uses 64 bits instructions, but the userspace uses 32 bits.

get_unmapped_area called by the `mmap()` syscall, used when %!CONFIG_MMU

mmap operations needed to implement the `mmap()` syscall

open operations needed to implement the *open()* syscall

release operations needed to implement the `release()` syscall

Description

Note:

Those operations are used to implemente the fs struct file_operations at the V4L2 drivers. The V4L2 core overrides the fs ops with some extra logic needed by the subsystem.

struct **video_device**
Structure used to create and manage the V4L2 device nodes.

Definition

```
struct video_device {
#if defined(CONFIG_MEDIA_CONTROLLER)
  struct media_entity entity;
  struct media_intf_devnode * intf_devnode;
  struct media_pipeline pipe;
#endif
  const struct v4l2_file_operations * fops;
  u32 device_caps;
  struct device dev;
  struct cdev * cdev;
  struct v4l2_device * v4l2_dev;
  struct device * dev_parent;
  struct v4l2_ctrl_handler * ctrl_handler;
  struct vb2_queue * queue;
  struct v4l2_prio_state * prio;
  char name[32];
  int vfl_type;
  int vfl_dir;
  int minor;
  u16 num;
  unsigned long flags;
  int index;
  spinlock_t fh_lock;
  struct list_head fh_list;
  int dev_debug;
  v4l2_std_id tvnorms;
  void (* release) (struct video_device *vdev);
  const struct v4l2_ioctl_ops * ioctl_ops;
  unsigned long valid_ioctls[BITS_TO_LONGS(BASE_VIDIOC_PRIVATE)];
  unsigned long disable_locking[BITS_TO_LONGS(BASE_VIDIOC_PRIVATE)];
  struct mutex * lock;
};
```

Members

entity *struct media_entity*

intf_devnode pointer to *struct media_intf_devnode*

pipe *struct media_pipeline*

fops pointer to *struct v4l2_file_operations* for the video device

device_caps device capabilities as used in v4l2_capabilities

dev `struct device` for the video device

cdev character device

v4l2_dev pointer to *struct v4l2_device* parent

dev_parent pointer to `struct device` parent

ctrl_handler Control handler associated with this device node. May be NULL.

queue *struct vb2_queue* associated with this device node. May be NULL.

prio pointer to *struct v4l2_prio_state* with device's Priority state. If NULL, then v4l2_dev->prio will be used.

name[32] video device name

vfl_type V4L device type

vfl_dir V4L receiver, transmitter or m2m

minor device node 'minor'. It is set to -1 if the registration failed

num number of the video device node

flags video device flags. Use bitops to set/clear/test flags

index attribute to differentiate multiple indices on one physical device

fh_lock Lock for all v4l2_fhs

fh_list List of *struct v4l2_fh*

dev_debug Internal device debug flags, not for use by drivers

tvnorms Supported tv norms

release video device `release()` callback

ioctl_ops pointer to *struct v4l2_ioctl_ops* with ioctl callbacks

valid_ioctls[BITS_TO_LONGS(BASE_VIDIOC_PRIVATE)] bitmap with the valid ioctls for this device

disable_locking[BITS_TO_LONGS(BASE_VIDIOC_PRIVATE)] bitmap with the ioctls that don't require locking

lock pointer to `struct mutex` serialization lock

Description

Note:

*Only set **dev_parent** if that can't be deduced from **v4l2_dev**.*

int **__video_register_device**(struct *video_device* * *vdev*, int *type*, int *nr*, int *warn_if_nr_in_use*, struct module * *owner*)
register video4linux devices

Parameters

struct video_device * vdev struct video_device to register

int type type of device to register

int nr which device node number is desired: (0 == /dev/video0, 1 == /dev/video1, ..., -1 == first free)

int warn_if_nr_in_use warn if the desired device node number was already in use and another number was chosen instead.

struct module * owner module that owns the video device node

Description

The registration code assigns minor numbers and device node numbers based on the requested type and registers the new device node with the kernel.

This function assumes that struct video_device was zeroed when it was allocated and does not contain any stale date.

An error is returned if no free minor or device node number could be found, or if the registration of the device node failed.

Returns 0 on success.

Valid values for **type** are:

- VFL_TYPE_GRABBER - A frame grabber
- VFL_TYPE_VBI - Vertical blank data (undecoded)
- VFL_TYPE_RADIO - A radio card
- VFL_TYPE_SUBDEV - A subdevice
- VFL_TYPE_SDR - Software Defined Radio
- VFL_TYPE_TOUCH - A touch sensor

Note:

This function is meant to be used only inside the V4L2 core. Drivers should use `video_register_device()` *or* `video_register_device_no_warn()`*.*

int **video_register_device**(struct *video_device* * *vdev*, int *type*, int *nr*)
register video4linux devices

Parameters

struct video_device * vdev struct video_device to register

int type type of device to register

int nr which device node number is desired: (0 == /dev/video0, 1 == /dev/video1, ..., -1 == first free)

Description

Internally, it calls *__video_register_device()*. Please see its documentation for more details.

Note:

if video_register_device fails, the `release()` *callback of* `struct video_device` *structure is* not *called, so the caller is responsible for freeing any data. Usually that means that you* `video_device_release()` *should be called on failure.*

int **video_register_device_no_warn**(struct *video_device* * *vdev*, int *type*, int *nr*)
register video4linux devices

Parameters

struct video_device * vdev struct video_device to register

int type type of device to register

int nr which device node number is desired: (0 == /dev/video0, 1 == /dev/video1, ..., -1 == first free)

Description

This function is identical to *video_register_device()* except that no warning is issued if the desired device node number was already in use.

Internally, it calls *__video_register_device()*. Please see its documentation for more details.

Note:

if video_register_device fails, the `release()` *callback of* `struct video_device` *structure is* not *called, so the caller is responsible for freeing any data. Usually that means that you* `video_device_release()` *should be called on failure.*

void **video_unregister_device**(struct *video_device* * *vdev*)
Unregister video devices.

Parameters

struct video_device * vdev *struct video_device* to register

Description

Does nothing if vdev == NULL or if *video_is_registered()* returns false.

struct *video_device* * **video_device_alloc**(void)
helper function to alloc *struct video_device*

Parameters

void no arguments

Description

Returns NULL if -ENOMEM or a *struct video_device* on success.

void **video_device_release**(struct *video_device* * *vdev*)
helper function to release *struct video_device*

Parameters

struct video_device * vdev pointer to *struct video_device*

Description

Can also be used for video_device->release().

void **video_device_release_empty**(struct *video_device* * *vdev*)
helper function to implement the video_device->release() callback.

Parameters

struct video_device * vdev pointer to *struct video_device*

Description

This release function does nothing.

It should be used when the video_device is a static global struct.

Note:

Having a static video_device is a dubious construction at best.

bool **v4l2_is_known_ioctl**(unsigned int *cmd*)
Checks if a given cmd is a known V4L ioctl

Parameters

unsigned int cmd ioctl command

Description

returns true if cmd is a known V4L2 ioctl

void **v4l2_disable_ioctl**(struct *video_device* * *vdev*, unsigned int *cmd*)
mark that a given command isn't implemented. shouldn't use core locking

Parameters

struct video_device * vdev pointer to *struct video_device*

unsigned int cmd ioctl command

Description

This function allows drivers to provide just one v4l2_ioctl_ops struct, but disable ioctls based on the specific card that is actually found.

Note:

This must be called before video_register_device. See also the comments for determine_valid_ioctls().

void * **video_get_drvdata**(struct *video_device* * *vdev*)
gets private data from *struct video_device*.

Parameters

struct video_device * vdev pointer to *struct video_device*

Description

returns a pointer to the private data

void **video_set_drvdata**(struct *video_device* * *vdev*, void * *data*)
sets private data from *struct video_device*.

Parameters

struct video_device * vdev pointer to *struct video_device*

void * data private data pointer

struct *video_device* * **video_devdata**(struct file * *file*)
gets *struct video_device* from struct file.

Parameters

struct file * file pointer to struct file

void * **video_drvdata**(struct file * *file*)
gets private data from *struct video_device* using the struct file.

Parameters

struct file * file pointer to struct file

Description

This is function combines both *video_get_drvdata()* and *video_devdata()* as this is used very often.

const char * **video_device_node_name**(struct *video_device* * *vdev*)
returns the video device name

Parameters

struct video_device * vdev pointer to *struct video_device*

Description

Returns the device name string

int **video_is_registered**(struct *video_device* * *vdev*)
returns true if the *struct video_device* is registered.

Parameters

struct video_device * vdev pointer to *struct video_device*

Description

2.1.5 V4L2 device instance

Each device instance is represented by a struct *v4l2_device*. Very simple devices can just allocate this struct, but most of the time you would embed this struct inside a larger struct.

You must register the device instance by calling:

v4l2_device_register (dev, *v4l2_dev*).

Registration will initialize the *v4l2_device* struct. If the dev->driver_data field is NULL, it will be linked to *v4l2_dev* argument.

Drivers that want integration with the media device framework need to set dev->driver_data manually to point to the driver-specific device structure that embed the struct *v4l2_device* instance. This is achieved by a `dev_set_drvdata()` call before registering the V4L2 device instance. They must also set the struct *v4l2_device* mdev field to point to a properly initialized and registered *media_device* instance.

If *v4l2_dev*->name is empty then it will be set to a value derived from dev (driver name followed by the bus_id, to be precise). If you set it up before calling *v4l2_device_register()* then it will be untouched. If dev is NULL, then you **must** setup *v4l2_dev*->name before calling *v4l2_device_register()*.

You can use *v4l2_device_set_name()* to set the name based on a driver name and a driver-global atomic_t instance. This will generate names like `ivtv0`, `ivtv1`, etc. If the name ends with a digit, then it will insert a dash: `cx18-0`, `cx18-1`, etc. This function returns the instance number.

The first dev argument is normally the `struct device` pointer of a `pci_dev`, `usb_interface` or `platform_device`. It is rare for dev to be NULL, but it happens with ISA devices or when one device creates multiple PCI devices, thus making it impossible to associate *v4l2_dev* with a particular parent.

You can also supply a `notify()` callback that can be called by sub-devices to notify you of events. Whether you need to set this depends on the sub-device. Any notifications a sub-device supports must be defined in a header in `include/media/subdevice.h`.

V4L2 devices are unregistered by calling:

v4l2_device_unregister() (*v4l2_dev*).

If the dev->driver_data field points to *v4l2_dev*, it will be reset to NULL. Unregistering will also automatically unregister all subdevs from the device.

If you have a hotpluggable device (e.g. a USB device), then when a disconnect happens the parent device becomes invalid. Since *v4l2_device* has a pointer to that parent device it has to be cleared as well to mark that the parent is gone. To do this call:

v4l2_device_disconnect() (*v4l2_dev*).

This does *not* unregister the subdevs, so you still need to call the *v4l2_device_unregister()* function for that. If your driver is not hotpluggable, then there is no need to call *v4l2_device_disconnect()*.

Sometimes you need to iterate over all devices registered by a specific driver. This is usually the case if multiple device drivers use the same hardware. E.g. the ivtvfb driver is a framebuffer driver that uses the ivtv hardware. The same is true for alsa drivers for example.

You can iterate over all registered devices as follows:

```
static int callback(struct device *dev, void *p)
{
        struct v4l2_device *v4l2_dev = dev_get_drvdata(dev);

        /* test if this device was inited */
        if (v4l2_dev == NULL)
```

```
                return 0;
        ...
        return 0;
}

int iterate(void *p)
{
        struct device_driver *drv;
        int err;

        /* Find driver 'ivtv' on the PCI bus.
        pci_bus_type is a global. For USB busses use usb_bus_type. */
        drv = driver_find("ivtv", &pci_bus_type);
        /* iterate over all ivtv device instances */
        err = driver_for_each_device(drv, NULL, p, callback);
        put_driver(drv);
        return err;
}
```

Sometimes you need to keep a running counter of the device instance. This is commonly used to map a device instance to an index of a module option array.

The recommended approach is as follows:

```
static atomic_t drv_instance = ATOMIC_INIT(0);

static int drv_probe(struct pci_dev *pdev, const struct pci_device_id *pci_id)
{
        ...
        state->instance = atomic_inc_return(&drv_instance) - 1;
}
```

If you have multiple device nodes then it can be difficult to know when it is safe to unregister *v4l2_device* for hotpluggable devices. For this purpose *v4l2_device* has refcounting support. The refcount is increased whenever *video_register_device()* is called and it is decreased whenever that device node is released. When the refcount reaches zero, then the *v4l2_device* release() callback is called. You can do your final cleanup there.

If other device nodes (e.g. ALSA) are created, then you can increase and decrease the refcount manually as well by calling:

v4l2_device_get() (*v4l2_dev*).

or:

v4l2_device_put() (*v4l2_dev*).

Since the initial refcount is 1 you also need to call *v4l2_device_put()* in the `disconnect()` callback (for USB devices) or in the `remove()` callback (for e.g. PCI devices), otherwise the refcount will never reach 0.

v4l2_device functions and data structures

struct **v4l2_device**
main struct to for V4L2 device drivers

Definition

```
struct v4l2_device {
  struct device * dev;
#if defined(CONFIG_MEDIA_CONTROLLER)
  struct media_device * mdev;
#endif
  struct list_head subdevs;
```

```
  spinlock_t lock;
  char name[V4L2_DEVICE_NAME_SIZE];
  void (* notify) (struct v4l2_subdev *sd,unsigned int notification, void *arg);
  struct v4l2_ctrl_handler * ctrl_handler;
  struct v4l2_prio_state prio;
  struct kref ref;
  void (* release) (struct v4l2_device *v4l2_dev);
};
```

Members

dev pointer to struct device.

mdev pointer to struct media_device

subdevs used to keep track of the registered subdevs

lock lock this struct; can be used by the driver as well if this struct is embedded into a larger struct.

name[V4L2_DEVICE_NAME_SIZE] unique device name, by default the driver name + bus ID

notify notify callback called by some sub-devices.

ctrl_handler The control handler. May be NULL.

prio Device's priority state

ref Keep track of the references to this struct.

release Release function that is called when the ref count goes to 0.

Description

Each instance of a V4L2 device should create the v4l2_device struct, either stand-alone or embedded in a larger struct.

It allows easy access to sub-devices (see v4l2-subdev.h) and provides basic V4L2 device-level support.

Note:

1. ***dev*->*driver_data points to this struct.*
2. ***dev** might be NULL if there is no parent device*

void **v4l2_device_get**(struct *v4l2_device* * *v4l2_dev*)
gets a V4L2 device reference

Parameters

struct v4l2_device * v4l2_dev pointer to struct *v4l2_device*

Description

This is an ancillary routine meant to increment the usage for the struct *v4l2_device* pointed by **v4l2_dev**.

int **v4l2_device_put**(struct *v4l2_device* * *v4l2_dev*)
putss a V4L2 device reference

Parameters

struct v4l2_device * v4l2_dev pointer to struct *v4l2_device*

Description

This is an ancillary routine meant to decrement the usage for the struct *v4l2_device* pointed by **v4l2_dev**.

int **v4l2_device_register**(struct device * *dev*, struct *v4l2_device* * *v4l2_dev*)
Initialize v4l2_dev and make **dev**->driver_data point to **v4l2_dev**.

Parameters

struct device * dev pointer to struct device

struct v4l2_device * v4l2_dev pointer to struct *v4l2_device*

Description

Note:

dev *may be NULL in rare cases (ISA devices). In such case the caller must fill in the* ***v4l2_dev****->name field before calling this function.*

int **v4l2_device_set_name**(struct *v4l2_device* * *v4l2_dev*, const char * *basename*, atomic_t * *instance*)
Optional function to initialize the name field of struct *v4l2_device*

Parameters

struct v4l2_device * v4l2_dev pointer to struct *v4l2_device*

const char * basename base name for the device name

atomic_t * instance pointer to a static atomic_t var with the instance usage for the device driver.

Description

v4l2_device_set_name() initializes the name field of struct *v4l2_device* using the driver name and a driver-global atomic_t instance.

This function will increment the instance counter and returns the instance value used in the name.

Example

static atomic_t drv_instance = ATOMIC_INIT(0);

...

instance = v4l2_device_set_name(&v4l2_dev, “foo”, &drv_instance);

The first time this is called the name field will be set to foo0 and this function returns 0. If the name ends with a digit (e.g. cx18), then the name will be set to cx18-0 since cx180 would look really odd.

void **v4l2_device_disconnect**(struct *v4l2_device* * *v4l2_dev*)
Change V4L2 device state to disconnected.

Parameters

struct v4l2_device * v4l2_dev pointer to struct v4l2_device

Description

Should be called when the USB parent disconnects. Since the parent disappears, this ensures that **v4l2_dev** doesn’t have an invalid parent pointer.

Note:

This function sets ***v4l2_dev****->dev to NULL.*

void **v4l2_device_unregister**(struct *v4l2_device* * *v4l2_dev*)
Unregister all sub-devices and any other resources related to **v4l2_dev**.

Parameters

struct v4l2_device * v4l2_dev pointer to struct v4l2_device

int **v4l2_device_register_subdev**(struct *v4l2_device* * *v4l2_dev*, struct *v4l2_subdev* * *sd*)
Registers a subdev with a v4l2 device.

Parameters

struct v4l2_device * v4l2_dev pointer to struct *v4l2_device*

struct v4l2_subdev * sd pointer to struct *v4l2_subdev*

Description

While registered, the subdev module is marked as in-use.

An error is returned if the module is no longer loaded on any attempts to register it.

void **v4l2_device_unregister_subdev**(struct *v4l2_subdev* * *sd*)
Unregisters a subdev with a v4l2 device.

Parameters

struct v4l2_subdev * sd pointer to struct *v4l2_subdev*

Description

Note:

Can also be called if the subdev wasn't registered. In such case, it will do nothing.

int **v4l2_device_register_subdev_nodes**(struct *v4l2_device* * *v4l2_dev*)
Registers device nodes for all subdevs of the v4l2 device that are marked with the V4L2_SUBDEV_FL_HAS_DEVNODE flag.

Parameters

struct v4l2_device * v4l2_dev pointer to struct v4l2_device

void **v4l2_subdev_notify**(struct *v4l2_subdev* * *sd*, unsigned int *notification*, void * *arg*)
Sends a notification to v4l2_device.

Parameters

struct v4l2_subdev * sd pointer to struct *v4l2_subdev*

unsigned int notification type of notification. Please notice that the notification type is driver-specific.

void * arg arguments for the notification. Those are specific to each notification type.

2.1.6 V4L2 File handlers

struct *v4l2_fh* provides a way to easily keep file handle specific data that is used by the V4L2 framework.

Attention:

New drivers must use struct v4l2_fh *since it is also used to implement priority handling (* ioctl VIDIOC_G_PRIORITY, VIDIOC_S_PRIORITY *).*

The users of *v4l2_fh* (in the V4L2 framework, not the driver) know whether a driver uses *v4l2_fh* as its `file->private_data` pointer by testing the V4L2_FL_USES_V4L2_FH bit in *video_device*->flags. This bit is set whenever *v4l2_fh_init()* is called.

struct *v4l2_fh* is allocated as a part of the driver's own file handle structure and `file->private_data` is set to it in the driver's `open()` function by the driver.

In many cases the struct *v4l2_fh* will be embedded in a larger structure. In that case you should call:

1. *v4l2_fh_init()* and *v4l2_fh_add()* in open()
2. *v4l2_fh_del()* and *v4l2_fh_exit()* in release()

Drivers can extract their own file handle structure by using the container_of macro.

Example:

```
struct my_fh {
        int blah;
        struct v4l2_fh fh;
};

...

int my_open(struct file *file)
{
        struct my_fh *my_fh;
        struct video_device *vfd;
        int ret;

        ...

        my_fh = kzalloc(sizeof(*my_fh), GFP_KERNEL);

        ...

        v4l2_fh_init(&my_fh->fh, vfd);

        ...

        file->private_data = &my_fh->fh;
        v4l2_fh_add(&my_fh->fh);
        return 0;
}

int my_release(struct file *file)
{
        struct v4l2_fh *fh = file->private_data;
        struct my_fh *my_fh = container_of(fh, struct my_fh, fh);

        ...
        v4l2_fh_del(&my_fh->fh);
        v4l2_fh_exit(&my_fh->fh);
        kfree(my_fh);
        return 0;
}
```

Below is a short description of the *v4l2_fh* functions used:

v4l2_fh_init (*fh*, *vdev*)

- Initialise the file handle. This **MUST** be performed in the driver's *v4l2_file_operations*->open() handler.

v4l2_fh_add (*fh*)

- Add a *v4l2_fh* to *video_device* file handle list. Must be called once the file handle is completely initialized.

v4l2_fh_del (*fh*)

- Unassociate the file handle from *video_device*. The file handle exit function may now be called.

v4l2_fh_exit (*fh*)

- Uninitialise the file handle. After uninitialisation the *v4l2_fh* memory can be freed.

If struct *v4l2_fh* is not embedded, then you can use these helper functions:

v4l2_fh_open (struct file *filp)

- This allocates a struct *v4l2_fh*, initializes it and adds it to the struct *video_device* associated with the file struct.

v4l2_fh_release (struct file *filp)

- This deletes it from the struct *video_device* associated with the file struct, uninitialised the *v4l2_fh* and frees it.

These two functions can be plugged into the v4l2_file_operation's `open()` and `release()` ops.

Several drivers need to do something when the first file handle is opened and when the last file handle closes. Two helper functions were added to check whether the *v4l2_fh* struct is the only open filehandle of the associated device node:

v4l2_fh_is_singular (*fh*)

- Returns 1 if the file handle is the only open file handle, else 0.

v4l2_fh_is_singular_file (struct file *filp)

- Same, but it calls v4l2_fh_is_singular with filp->private_data.

V4L2 fh functions and data structures

struct **v4l2_fh**
Describes a V4L2 file handler

Definition

```
struct v4l2_fh {
  struct list_head list;
  struct video_device * vdev;
  struct v4l2_ctrl_handler * ctrl_handler;
  enum v4l2_priority prio;
  wait_queue_head_t wait;
  struct list_head subscribed;
  struct list_head available;
  unsigned int navailable;
  u32 sequence;
#if IS_ENABLED(CONFIG_V4L2_MEM2MEM_DEV)
  struct v4l2_m2m_ctx * m2m_ctx;
#endif
};
```

Members

list list of file handlers

vdev pointer to *struct video_device*

ctrl_handler pointer to *struct v4l2_ctrl_handler*

prio priority of the file handler, as defined by *enum v4l2_priority*

wait event' s wait queue

subscribed list of subscribed events

available list of events waiting to be dequeued

navailable number of available events at **available** list

sequence event sequence number

m2m_ctx pointer to *struct v4l2_m2m_ctx*

void **v4l2_fh_init**(struct *v4l2_fh* * *fh*, struct *video_device* * *vdev*)
Initialise the file handle.

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

struct video_device * vdev pointer to *struct video_device*

Description

Parts of the V4L2 framework using the file handles should be initialised in this function. Must be called from driver's v4l2_file_operations->open() handler if the driver uses *struct v4l2_fh*.

void **v4l2_fh_add**(struct *v4l2_fh* * *fh*)
Add the fh to the list of file handles on a video_device.

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

Description

Note:

*The **fh** file handle must be initialised first.*

int **v4l2_fh_open**(struct file * *filp*)
Ancillary routine that can be used as the open() op of v4l2_file_operations.

Parameters

struct file * filp pointer to struct file

Description

It allocates a v4l2_fh and inits and adds it to the *struct video_device* associated with the file pointer.

void **v4l2_fh_del**(struct *v4l2_fh* * *fh*)
Remove file handle from the list of file handles.

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

Description

On error filp->private_data will be NULL, otherwise it will point to the *struct v4l2_fh*.

Note:

Must be called in v4l2_file_operations->release() handler if the driver uses `struct v4l2_fh`.

void **v4l2_fh_exit**(struct *v4l2_fh* * *fh*)
Release resources related to a file handle.

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

Description

Parts of the V4L2 framework using the v4l2_fh must release their resources here, too.

Note:

Must be called in v4l2_file_operations->release() handler if the driver uses `struct v4l2_fh`.

int **v4l2_fh_release**(struct file * *filp*)
Ancillary routine that can be used as the release() op of v4l2_file_operations.

Parameters

`struct file * filp` pointer to struct file

Description

It deletes and exits the v4l2_fh associated with the file pointer and frees it. It will do nothing if filp->private_data (the pointer to the v4l2_fh struct) is NULL.

This function always returns 0.

int **v4l2_fh_is_singular**(struct *v4l2_fh* * *fh*)
Returns 1 if this filehandle is the only filehandle opened for the associated video_device.

Parameters

`struct v4l2_fh * fh` pointer to *`struct v4l2_fh`*

Description

If **fh** is NULL, then it returns 0.

int **v4l2_fh_is_singular_file**(struct file * *filp*)
Returns 1 if this filehandle is the only filehandle opened for the associated video_device.

Parameters

`struct file * filp` pointer to struct file

Description

This is a helper function variant of *`v4l2_fh_is_singular()`* with uses struct file as argument.

If filp->private_data is NULL, then it will return 0.

2.1.7 V4L2 sub-devices

Many drivers need to communicate with sub-devices. These devices can do all sort of tasks, but most commonly they handle audio and/or video muxing, encoding or decoding. For webcams common sub-devices are sensors and camera controllers.

Usually these are I2C devices, but not necessarily. In order to provide the driver with a consistent interface to these sub-devices the *v4l2_subdev* struct (v4l2-subdev.h) was created.

Each sub-device driver must have a *v4l2_subdev* struct. This struct can be stand-alone for simple sub-devices or it might be embedded in a larger struct if more state information needs to be stored. Usually there is a low-level device struct (e.g. `i2c_client`) that contains the device data as setup by the kernel. It is recommended to store that pointer in the private data of *v4l2_subdev* using *`v4l2_set_subdevdata()`*. That makes it easy to go from a *v4l2_subdev* to the actual low-level bus-specific device data.

You also need a way to go from the low-level struct to *v4l2_subdev*. For the common i2c_client struct the i2c_set_clientdata() call is used to store a *v4l2_subdev* pointer, for other busses you may have to use other methods.

Bridges might also need to store per-subdev private data, such as a pointer to bridge-specific per-subdev private data. The *v4l2_subdev* structure provides host private data for that purpose that can be accessed with *`v4l2_get_subdev_hostdata()`* and *`v4l2_set_subdev_hostdata()`*.

From the bridge driver perspective, you load the sub-device module and somehow obtain the *v4l2_subdev* pointer. For i2c devices this is easy: you call `i2c_get_clientdata()`. For other busses something similar needs to be done. Helper functions exists for sub-devices on an I2C bus that do most of this tricky work for you.

Each *v4l2_subdev* contains function pointers that sub-device drivers can implement (or leave NULL if it is not applicable). Since sub-devices can do so many different things and you do not want to end up with a huge ops struct of which only a handful of ops are commonly implemented, the function pointers are sorted according to category and each category has its own ops struct.

The top-level ops struct contains pointers to the category ops structs, which may be NULL if the subdev driver does not support anything from that category.

It looks like this:

```
struct v4l2_subdev_core_ops {
        int (*log_status)(struct v4l2_subdev *sd);
        int (*init)(struct v4l2_subdev *sd, u32 val);
        ...
};

struct v4l2_subdev_tuner_ops {
        ...
};

struct v4l2_subdev_audio_ops {
        ...
};

struct v4l2_subdev_video_ops {
        ...
};

struct v4l2_subdev_pad_ops {
        ...
};

struct v4l2_subdev_ops {
        const struct v4l2_subdev_core_ops  *core;
        const struct v4l2_subdev_tuner_ops *tuner;
        const struct v4l2_subdev_audio_ops *audio;
        const struct v4l2_subdev_video_ops *video;
        const struct v4l2_subdev_pad_ops *video;
};
```

The core ops are common to all subdevs, the other categories are implemented depending on the sub-device. E.g. a video device is unlikely to support the audio ops and vice versa.

This setup limits the number of function pointers while still making it easy to add new ops and categories.

A sub-device driver initializes the *v4l2_subdev* struct using:

v4l2_subdev_init (*sd*, &*ops*).

Afterwards you need to initialize *sd*->name with a unique name and set the module owner. This is done for you if you use the i2c helper functions.

If integration with the media framework is needed, you must initialize the *media_entity* struct embedded in the *v4l2_subdev* struct (entity field) by calling *media_entity_pads_init()*, if the entity has pads:

```
struct media_pad *pads = &my_sd->pads;
int err;

err = media_entity_pads_init(&sd->entity, npads, pads);
```

The pads array must have been previously initialized. There is no need to manually set the struct *media_entity* function and name fields, but the revision field must be initialized if needed.

A reference to the entity will be automatically acquired/released when the subdev device node (if any) is opened/closed.

Don't forget to cleanup the media entity before the sub-device is destroyed:

```
media_entity_cleanup(&sd->entity);
```

If the subdev driver intends to process video and integrate with the media framework, it must implement format related functionality using *v4l2_subdev_pad_ops* instead of *v4l2_subdev_video_ops*.

In that case, the subdev driver may set the link_validate field to provide its own link validation function. The link validation function is called for every link in the pipeline where both of the ends of the links are V4L2 sub-devices. The driver is still responsible for validating the correctness of the format configuration between sub-devices and video nodes.

If link_validate op is not set, the default function *v4l2_subdev_link_validate_default()* is used instead. This function ensures that width, height and the media bus pixel code are equal on both source and sink of the link. Subdev drivers are also free to use this function to perform the checks mentioned above in addition to their own checks.

There are currently two ways to register subdevices with the V4L2 core. The first (traditional) possibility is to have subdevices registered by bridge drivers. This can be done when the bridge driver has the complete information about subdevices connected to it and knows exactly when to register them. This is typically the case for internal subdevices, like video data processing units within SoCs or complex PCI(e) boards, camera sensors in USB cameras or connected to SoCs, which pass information about them to bridge drivers, usually in their platform data.

There are however also situations where subdevices have to be registered asynchronously to bridge devices. An example of such a configuration is a Device Tree based system where information about subdevices is made available to the system independently from the bridge devices, e.g. when subdevices are defined in DT as I2C device nodes. The API used in this second case is described further below.

Using one or the other registration method only affects the probing process, the run-time bridge-subdevice interaction is in both cases the same.

In the synchronous case a device (bridge) driver needs to register the *v4l2_subdev* with the v4l2_device:

v4l2_device_register_subdev (*v4l2_dev*, *sd*).

This can fail if the subdev module disappeared before it could be registered. After this function was called successfully the subdev->dev field points to the *v4l2_device*.

If the v4l2_device parent device has a non-NULL mdev field, the sub-device entity will be automatically registered with the media device.

You can unregister a sub-device using:

v4l2_device_unregister_subdev (*sd*).

Afterwards the subdev module can be unloaded and *sd*->dev == NULL.

You can call an ops function either directly:

```
err = sd->ops->core->g_std(sd, &norm);
```

but it is better and easier to use this macro:

```
err = v4l2_subdev_call(sd, core, g_std, &norm);
```

The macro will to the right NULL pointer checks and returns -ENODEV if *sd* is NULL, -ENOIOCTLCMD if either *sd*->core or *sd*->core->g_std is NULL, or the actual result of the *sd*->ops->core->g_std ops.

It is also possible to call all or a subset of the sub-devices:

```
v4l2_device_call_all(v4l2_dev, 0, core, g_std, &norm);
```

Any subdev that does not support this ops is skipped and error results are ignored. If you want to check for errors use this:

```
err = v4l2_device_call_until_err(v4l2_dev, 0, core, g_std, &norm);
```

Any error except -ENOIOCTLCMD will exit the loop with that error. If no errors (except -ENOIOCTLCMD) occurred, then 0 is returned.

The second argument to both calls is a group ID. If 0, then all subdevs are called. If non-zero, then only those whose group ID match that value will be called. Before a bridge driver registers a subdev it can set *sd*->grp_id to whatever value it wants (it's 0 by default). This value is owned by the bridge driver and the sub-device driver will never modify or use it.

The group ID gives the bridge driver more control how callbacks are called. For example, there may be multiple audio chips on a board, each capable of changing the volume. But usually only one will actually be used when the user want to change the volume. You can set the group ID for that subdev to e.g. AUDIO_CONTROLLER and specify that as the group ID value when calling `v4l2_device_call_all()`. That ensures that it will only go to the subdev that needs it.

If the sub-device needs to notify its v4l2_device parent of an event, then it can call `v4l2_subdev_notify(sd,notification,arg)`. This macro checks whether there is a `notify()` callback defined and returns -ENODEV if not. Otherwise the result of the `notify()` call is returned.

The advantage of using *v4l2_subdev* is that it is a generic struct and does not contain any knowledge about the underlying hardware. So a driver might contain several subdevs that use an I2C bus, but also a subdev that is controlled through GPIO pins. This distinction is only relevant when setting up the device, but once the subdev is registered it is completely transparent.

In the asynchronous case subdevice probing can be invoked independently of the bridge driver availability. The subdevice driver then has to verify whether all the requirements for a successful probing are satisfied. This can include a check for a master clock availability. If any of the conditions aren't satisfied the driver might decide to return -EPROBE_DEFER to request further reprobing attempts. Once all conditions are met the subdevice shall be registered using the *v4l2_async_register_subdev()* function. Unregistration is performed using the *v4l2_async_unregister_subdev()* call. Subdevices registered this way are stored in a global list of subdevices, ready to be picked up by bridge drivers.

Bridge drivers in turn have to register a notifier object with an array of subdevice descriptors that the bridge device needs for its operation. This is performed using the *v4l2_async_notifier_register()* call. To unregister the notifier the driver has to call *v4l2_async_notifier_unregister()*. The former of the two functions takes two arguments: a pointer to struct *v4l2_device* and a pointer to struct *v4l2_async_notifier*. The latter contains a pointer to an array of pointers to subdevice descriptors of type struct *v4l2_async_subdev* type. The V4L2 core will then use these descriptors to match asynchronously registered subdevices to them. If a match is detected the `.bound()` notifier callback is called. After all subdevices have been located the .complete() callback is called. When a subdevice is removed from the system the .unbind() method is called. All three callbacks are optional.

2.1.8 V4L2 sub-device userspace API

Beside exposing a kernel API through the *v4l2_subdev_ops* structure, V4L2 sub-devices can also be controlled directly by userspace applications.

Device nodes named `v4l-subdev`*X* can be created in `/dev` to access sub-devices directly. If a sub-device supports direct userspace configuration it must set the V4L2_SUBDEV_FL_HAS_DEVNODE flag before being registered.

After registering sub-devices, the *v4l2_device* driver can create device nodes for all registered sub-devices marked with V4L2_SUBDEV_FL_HAS_DEVNODE by calling *v4l2_device_register_subdev_nodes()*. Those device nodes will be automatically removed when sub-devices are unregistered.

The device node handles a subset of the V4L2 API.

VIDIOC_QUERYCTRL, VIDIOC_QUERYMENU, VIDIOC_G_CTRL, VIDIOC_S_CTRL, VIDIOC_G_EXT_CTRLS, VIDIOC_S_EXT_CTRLS and VIDIOC_TRY_EXT_CTRLS:

The controls ioctls are identical to the ones defined in V4L2. They behave identically, with the only exception that they deal only with controls implemented in the sub-device. Depending on the driver, those controls can be also be accessed through one (or several) V4L2 device nodes.

VIDIOC_DQEVENT, VIDIOC_SUBSCRIBE_EVENT and VIDIOC_UNSUBSCRIBE_EVENT

The events ioctls are identical to the ones defined in V4L2. They behave identically, with the only exception that they deal only with events generated by the sub-device. Depending on the driver, those events can also be reported by one (or several) V4L2 device nodes.

Sub-device drivers that want to use events need to set the V4L2_SUBDEV_USES_EVENTS *v4l2_subdev*.flags and initialize *v4l2_subdev*.nevents to events queue depth before registering the sub-device. After registration events can be queued as usual on the *v4l2_subdev*.devnode device node.

To properly support events, the `poll()` file operation is also implemented.

Private ioctls

All ioctls not in the above list are passed directly to the sub-device driver through the core::ioctl operation.

2.1.9 I2C sub-device drivers

Since these drivers are so common, special helper functions are available to ease the use of these drivers (`v4l2-common.h`).

The recommended method of adding *v4l2_subdev* support to an I2C driver is to embed the *v4l2_subdev* struct into the state struct that is created for each I2C device instance. Very simple devices have no state struct and in that case you can just create a *v4l2_subdev* directly.

A typical state struct would look like this (where 'chipname' is replaced by the name of the chip):

```
struct chipname_state {
        struct v4l2_subdev sd;
        ...  /* additional state fields */
};
```

Initialize the *v4l2_subdev* struct as follows:

```
v4l2_i2c_subdev_init(&state->sd, client, subdev_ops);
```

This function will fill in all the fields of *v4l2_subdev* ensure that the *v4l2_subdev* and i2c_client both point to one another.

You should also add a helper inline function to go from a *v4l2_subdev* pointer to a chipname_state struct:

```
static inline struct chipname_state *to_state(struct v4l2_subdev *sd)
{
        return container_of(sd, struct chipname_state, sd);
}
```

Use this to go from the *v4l2_subdev* struct to the `i2c_client` struct:

```
struct i2c_client *client = v4l2_get_subdevdata(sd);
```

And this to go from an `i2c_client` to a *v4l2_subdev* struct:

```
struct v4l2_subdev *sd = i2c_get_clientdata(client);
```

Make sure to call *v4l2_device_unregister_subdev()*(*sd*) when the remove() callback is called. This will unregister the sub-device from the bridge driver. It is safe to call this even if the sub-device was never registered.

You need to do this because when the bridge driver destroys the i2c adapter the remove() callbacks are called of the i2c devices on that adapter. After that the corresponding v4l2_subdev structures are invalid, so they have to be unregistered first. Calling *v4l2_device_unregister_subdev()*(*sd*) from the remove() callback ensures that this is always done correctly.

The bridge driver also has some helper functions it can use:

```
struct v4l2_subdev *sd = v4l2_i2c_new_subdev(v4l2_dev, adapter,
                            "module_foo", "chipid", 0x36, NULL);
```

This loads the given module (can be NULL if no module needs to be loaded) and calls i2c_new_device() with the given i2c_adapter and chip/address arguments. If all goes well, then it registers the subdev with the v4l2_device.

You can also use the last argument of *v4l2_i2c_new_subdev()* to pass an array of possible I2C addresses that it should probe. These probe addresses are only used if the previous argument is 0. A non-zero argument means that you know the exact i2c address so in that case no probing will take place.

Both functions return NULL if something went wrong.

Note that the chipid you pass to *v4l2_i2c_new_subdev()* is usually the same as the module name. It allows you to specify a chip variant, e.g. “saa7114” or “saa7115”. In general though the i2c driver autodetects this. The use of chipid is something that needs to be looked at more closely at a later date. It differs between i2c drivers and as such can be confusing. To see which chip variants are supported you can look in the i2c driver code for the i2c_device_id table. This lists all the possibilities.

There are one more helper function:

v4l2_i2c_new_subdev_board() uses an i2c_board_info struct which is passed to the i2c driver and replaces the irq, platform_data and addr arguments.

If the subdev supports the s_config core ops, then that op is called with the irq and platform_data arguments after the subdev was setup.

The *v4l2_i2c_new_subdev()* function will call *v4l2_i2c_new_subdev_board()*, internally filling a i2c_board_info structure using the client_type and the addr to fill it.

2.1.10 V4L2 sub-device functions and data structures

struct **v4l2_decode_vbi_line**
used to decode_vbi_line

Definition

```
struct v4l2_decode_vbi_line {
  u32 is_second_field;
  u8 * p;
  u32 line;
  u32 type;
};
```

Members

is_second_field Set to 0 for the first (odd) field; set to 1 for the second (even) field.

p Pointer to the sliced VBI data from the decoder. On exit, points to the start of the payload.

line Line number of the sliced VBI data (1-23)

type VBI service type (V4L2_SLICED_*). 0 if no service found

struct **v4l2_subdev_io_pin_config**
Subdevice external IO pin configuration

Definition

```
struct v4l2_subdev_io_pin_config {
  u32 flags;
  u8 pin;
  u8 function;
  u8 value;
  u8 strength;
};
```

Members

flags bitmask with flags for this pin's config: V4L2_SUBDEV_IO_PIN_DISABLE - disables a pin config, V4L2_SUBDEV_IO_PIN_OUTPUT - if pin is an output, V4L2_SUBDEV_IO_PIN_INPUT - if pin is an input, V4L2_SUBDEV_IO_PIN_SET_VALUE - to set the output value via **value** and V4L2_SUBDEV_IO_PIN_ACTIVE_LOW - if active is 0.

pin Chip external IO pin to configure

function Internal signal pad/function to route to IO pin

value Initial value for pin - e.g. GPIO output value

strength Pin drive strength

struct **v4l2_subdev_core_ops**
Define core ops callbacks for subdevs

Definition

```
struct v4l2_subdev_core_ops {
  int (* log_status) (struct v4l2_subdev *sd);
  int (* s_io_pin_config) (struct v4l2_subdev *sd, size_t n,struct v4l2_subdev_io_pin_config␣
 ↪*pincfg);
  int (* init) (struct v4l2_subdev *sd, u32 val);
  int (* load_fw) (struct v4l2_subdev *sd);
  int (* reset) (struct v4l2_subdev *sd, u32 val);
  int (* s_gpio) (struct v4l2_subdev *sd, u32 val);
  long (* ioctl) (struct v4l2_subdev *sd, unsigned int cmd, void *arg);
#ifdef CONFIG_COMPAT
  long (* compat_ioctl32) (struct v4l2_subdev *sd, unsigned int cmd,unsigned long arg);
#endif
#ifdef CONFIG_VIDEO_ADV_DEBUG
  int (* g_register) (struct v4l2_subdev *sd, struct v4l2_dbg_register *reg);
  int (* s_register) (struct v4l2_subdev *sd, const struct v4l2_dbg_register *reg);
#endif
  int (* s_power) (struct v4l2_subdev *sd, int on);
  int (* interrupt_service_routine) (struct v4l2_subdev *sd,u32 status, bool *handled);
  int (* subscribe_event) (struct v4l2_subdev *sd, struct v4l2_fh *fh,struct v4l2_event_
 ↪subscription *sub);
  int (* unsubscribe_event) (struct v4l2_subdev *sd, struct v4l2_fh *fh,struct v4l2_event_
 ↪subscription *sub);
};
```

Members

log_status callback for VIDIOC_LOG_STATUS ioctl handler code.

s_io_pin_config configure one or more chip I/O pins for chips that multiplex different internal signal pads out to IO pins. This function takes a pointer to an array of 'n' pin configuration entries, one for each pin being configured. This function could be called at times other than just subdevice initialization.

init initialize the sensor registers to some sort of reasonable default values. Do not use for new drivers and should be removed in existing drivers.

load_fw load firmware.

reset generic reset command. The argument selects which subsystems to reset. Passing 0 will always reset the whole chip. Do not use for new drivers without discussing this first on the linux-media mailinglist. There should be no reason normally to reset a device.

s_gpio set GPIO pins. Very simple right now, might need to be extended with a direction argument if needed.

ioctl called at the end of `ioctl()` syscall handler at the V4L2 core. used to provide support for private ioctls used on the driver.

compat_ioctl32 called when a 32 bits application uses a 64 bits Kernel, in order to fix data passed from/to userspace.

g_register callback for `VIDIOC_G_REGISTER` ioctl handler code.

s_register callback for `VIDIOC_G_REGISTER` ioctl handler code.

s_power puts subdevice in power saving mode (on == 0) or normal operation mode (on == 1).

interrupt_service_routine Called by the bridge chip's interrupt service handler, when an interrupt status has be raised due to this subdev, so that this subdev can handle the details. It may schedule work to be performed later. It must not sleep. **Called from an IRQ context**.

subscribe_event used by the drivers to request the control framework that for it to be warned when the value of a control changes.

unsubscribe_event remove event subscription from the control framework.

struct **v4l2_subdev_tuner_ops**
Callbacks used when v4l device was opened in radio mode.

Definition

```
struct v4l2_subdev_tuner_ops {
  int (* s_radio) (struct v4l2_subdev *sd);
  int (* s_frequency) (struct v4l2_subdev *sd, const struct v4l2_frequency *freq);
  int (* g_frequency) (struct v4l2_subdev *sd, struct v4l2_frequency *freq);
  int (* enum_freq_bands) (struct v4l2_subdev *sd, struct v4l2_frequency_band *band);
  int (* g_tuner) (struct v4l2_subdev *sd, struct v4l2_tuner *vt);
  int (* s_tuner) (struct v4l2_subdev *sd, const struct v4l2_tuner *vt);
  int (* g_modulator) (struct v4l2_subdev *sd, struct v4l2_modulator *vm);
  int (* s_modulator) (struct v4l2_subdev *sd, const struct v4l2_modulator *vm);
  int (* s_type_addr) (struct v4l2_subdev *sd, struct tuner_setup *type);
  int (* s_config) (struct v4l2_subdev *sd, const struct v4l2_priv_tun_config *config);
};
```

Members

s_radio callback for `VIDIOC_S_RADIO` ioctl handler code.

s_frequency callback for `VIDIOC_S_FREQUENCY` ioctl handler code.

g_frequency callback for `VIDIOC_G_FREQUENCY` ioctl handler code. freq->type must be filled in. Normally done by *video_ioctl2()* or the bridge driver.

enum_freq_bands callback for `VIDIOC_ENUM_FREQ_BANDS` ioctl handler code.

g_tuner callback for `VIDIOC_G_TUNER` ioctl handler code.

s_tuner callback for `VIDIOC_S_TUNER` ioctl handler code. **vt**->type must be filled in. Normally done by video_ioctl2 or the bridge driver.

g_modulator callback for `VIDIOC_G_MODULATOR` ioctl handler code.

s_modulator callback for `VIDIOC_S_MODULATOR` ioctl handler code.

s_type_addr sets tuner type and its I2C addr.

s_config sets tda9887 specific stuff, like port1, port2 and qss

struct **v4l2_subdev_audio_ops**
Callbacks used for audio-related settings

Definition

```
struct v4l2_subdev_audio_ops {
  int (* s_clock_freq) (struct v4l2_subdev *sd, u32 freq);
  int (* s_i2s_clock_freq) (struct v4l2_subdev *sd, u32 freq);
  int (* s_routing) (struct v4l2_subdev *sd, u32 input, u32 output, u32 config);
  int (* s_stream) (struct v4l2_subdev *sd, int enable);
};
```

Members

s_clock_freq set the frequency (in Hz) of the audio clock output. Used to slave an audio processor to the video decoder, ensuring that audio and video remain synchronized. Usual values for the frequency are 48000, 44100 or 32000 Hz. If the frequency is not supported, then -EINVAL is returned.

s_i2s_clock_freq sets I2S speed in bps. This is used to provide a standard way to select I2S clock used by driving digital audio streams at some board designs. Usual values for the frequency are 1024000 and 2048000. If the frequency is not supported, then `-EINVAL` is returned.

s_routing used to define the input and/or output pins of an audio chip, and any additional configuration data. Never attempt to use user-level input IDs (e.g. Composite, S-Video, Tuner) at this level. An i2c device shouldn't know about whether an input pin is connected to a Composite connector, become on another board or platform it might be connected to something else entirely. The calling driver is responsible for mapping a user-level input to the right pins on the i2c device.

s_stream used to notify the audio code that stream will start or has stopped.

struct **v4l2_mbus_frame_desc_entry**
media bus frame description structure

Definition

```
struct v4l2_mbus_frame_desc_entry {
  u16 flags;
  u32 pixelcode;
  u32 length;
};
```

Members

flags bitmask flags: `V4L2_MBUS_FRAME_DESC_FL_LEN_MAX` and `V4L2_MBUS_FRAME_DESC_FL_BLOB`.

pixelcode media bus pixel code, valid if FRAME_DESC_FL_BLOB is not set

length number of octets per frame, valid if V4L2_MBUS_FRAME_DESC_FL_BLOB is set

struct **v4l2_mbus_frame_desc**
media bus data frame description

Definition

```
struct v4l2_mbus_frame_desc {
  struct v4l2_mbus_frame_desc_entry entry[V4L2_FRAME_DESC_ENTRY_MAX];
  unsigned short num_entries;
};
```

Members

entry[V4L2_FRAME_DESC_ENTRY_MAX] frame descriptors array

num_entries number of entries in **entry** array

struct **v4l2_subdev_video_ops**
Callbacks used when v4l device was opened in video mode.

Definition

```
struct v4l2_subdev_video_ops {
  int (* s_routing) (struct v4l2_subdev *sd, u32 input, u32 output, u32 config);
  int (* s_crystal_freq) (struct v4l2_subdev *sd, u32 freq, u32 flags);
  int (* g_std) (struct v4l2_subdev *sd, v4l2_std_id *norm);
  int (* s_std) (struct v4l2_subdev *sd, v4l2_std_id norm);
  int (* s_std_output) (struct v4l2_subdev *sd, v4l2_std_id std);
  int (* g_std_output) (struct v4l2_subdev *sd, v4l2_std_id *std);
  int (* querystd) (struct v4l2_subdev *sd, v4l2_std_id *std);
  int (* g_tvnorms) (struct v4l2_subdev *sd, v4l2_std_id *std);
  int (* g_tvnorms_output) (struct v4l2_subdev *sd, v4l2_std_id *std);
  int (* g_input_status) (struct v4l2_subdev *sd, u32 *status);
  int (* s_stream) (struct v4l2_subdev *sd, int enable);
  int (* g_pixelaspect) (struct v4l2_subdev *sd, struct v4l2_fract *aspect);
  int (* g_parm) (struct v4l2_subdev *sd, struct v4l2_streamparm *param);
  int (* s_parm) (struct v4l2_subdev *sd, struct v4l2_streamparm *param);
  int (* g_frame_interval) (struct v4l2_subdev *sd,struct v4l2_subdev_frame_interval *interval);
  int (* s_frame_interval) (struct v4l2_subdev *sd,struct v4l2_subdev_frame_interval *interval);
  int (* s_dv_timings) (struct v4l2_subdev *sd,struct v4l2_dv_timings *timings);
  int (* g_dv_timings) (struct v4l2_subdev *sd,struct v4l2_dv_timings *timings);
  int (* query_dv_timings) (struct v4l2_subdev *sd,struct v4l2_dv_timings *timings);
  int (* g_mbus_config) (struct v4l2_subdev *sd,struct v4l2_mbus_config *cfg);
  int (* s_mbus_config) (struct v4l2_subdev *sd,const struct v4l2_mbus_config *cfg);
  int (* s_rx_buffer) (struct v4l2_subdev *sd, void *buf,unsigned int *size);
};
```

Members

s_routing see s_routing in audio_ops, except this version is for video devices.

s_crystal_freq sets the frequency of the crystal used to generate the clocks in Hz. An extra flags field allows device specific configuration regarding clock frequency dividers, etc. If not used, then set flags to 0. If the frequency is not supported, then -EINVAL is returned.

g_std callback for VIDIOC_G_STD ioctl handler code.

s_std callback for VIDIOC_S_STD ioctl handler code.

s_std_output set v4l2_std_id for video OUTPUT devices. This is ignored by video input devices.

g_std_output get current standard for video OUTPUT devices. This is ignored by video input devices.

querystd callback for VIDIOC_QUERYSTD ioctl handler code.

g_tvnorms get v4l2_std_id with all standards supported by the video CAPTURE device. This is ignored by video output devices.

g_tvnorms_output get v4l2_std_id with all standards supported by the video OUTPUT device. This is ignored by video capture devices.

g_input_status get input status. Same as the status field in the struct *v4l2_input*

s_stream used to notify the driver that a video stream will start or has stopped.

g_pixelaspect callback to return the pixelaspect ratio.

g_parm callback for VIDIOC_G_PARM ioctl handler code.

s_parm callback for VIDIOC_S_PARM ioctl handler code.

g_frame_interval callback for VIDIOC_G_FRAMEINTERVAL ioctl handler code.

s_frame_interval callback for VIDIOC_S_FRAMEINTERVAL ioctl handler code.

s_dv_timings Set custom dv timings in the sub device. This is used when sub device is capable of setting detailed timing information in the hardware to generate/detect the video signal.

g_dv_timings Get custom dv timings in the sub device.

query_dv_timings callback for VIDIOC_QUERY_DV_TIMINGS ioctl handler code.

g_mbus_config get supported mediabus configurations

s_mbus_config set a certain mediabus configuration. This operation is added for compatibility with soc-camera drivers and should not be used by new software.

s_rx_buffer set a host allocated memory buffer for the subdev. The subdev can adjust **size** to a lower value and must not write more data to the buffer starting at **data** than the original value of **size**.

struct **v4l2_subdev_vbi_ops**
Callbacks used when v4l device was opened in video mode via the vbi device node.

Definition

```
struct v4l2_subdev_vbi_ops {
  int (* decode_vbi_line) (struct v4l2_subdev *sd, struct v4l2_decode_vbi_line *vbi_line);
  int (* s_vbi_data) (struct v4l2_subdev *sd, const struct v4l2_sliced_vbi_data *vbi_data);
  int (* g_vbi_data) (struct v4l2_subdev *sd, struct v4l2_sliced_vbi_data *vbi_data);
  int (* g_sliced_vbi_cap) (struct v4l2_subdev *sd, struct v4l2_sliced_vbi_cap *cap);
  int (* s_raw_fmt) (struct v4l2_subdev *sd, struct v4l2_vbi_format *fmt);
  int (* g_sliced_fmt) (struct v4l2_subdev *sd, struct v4l2_sliced_vbi_format *fmt);
  int (* s_sliced_fmt) (struct v4l2_subdev *sd, struct v4l2_sliced_vbi_format *fmt);
};
```

Members

decode_vbi_line video decoders that support sliced VBI need to implement this ioctl. Field p of the *struct v4l2_decode_vbi_line* is set to the start of the VBI data that was generated by the decoder. The driver then parses the sliced VBI data and sets the other fields in the struct accordingly. The pointer p is updated to point to the start of the payload which can be copied verbatim into the data field of the *struct v4l2_sliced_vbi_data*. If no valid VBI data was found, then the type field is set to 0 on return.

s_vbi_data used to generate VBI signals on a video signal. *struct v4l2_sliced_vbi_data* is filled with the data packets that should be output. Note that if you set the line field to 0, then that VBI signal is disabled. If no valid VBI data was found, then the type field is set to 0 on return.

g_vbi_data used to obtain the sliced VBI packet from a readback register. Not all video decoders support this. If no data is available because the readback register contains invalid or erroneous data -EI0 is returned. Note that you must fill in the 'id' member and the 'field' member (to determine whether CC data from the first or second field should be obtained).

g_sliced_vbi_cap callback for VIDIOC_SLICED_VBI_CAP ioctl handler code.

s_raw_fmt setup the video encoder/decoder for raw VBI.

g_sliced_fmt retrieve the current sliced VBI settings.

s_sliced_fmt setup the sliced VBI settings.

struct **v4l2_subdev_sensor_ops**
v4l2-subdev sensor operations

Definition

```
struct v4l2_subdev_sensor_ops {
  int (* g_skip_top_lines) (struct v4l2_subdev *sd, u32 *lines);
  int (* g_skip_frames) (struct v4l2_subdev *sd, u32 *frames);
};
```

Members

g_skip_top_lines number of lines at the top of the image to be skipped. This is needed for some sensors, which always corrupt several top lines of the output image, or which send their metadata in them.

g_skip_frames number of frames to skip at stream start. This is needed for buggy sensors that generate faulty frames when they are turned on.

enum **v4l2_subdev_ir_mode**
describes the type of IR supported

Constants

V4L2_SUBDEV_IR_MODE_PULSE_WIDTH IR uses struct ir_raw_event records

struct **v4l2_subdev_ir_parameters**
Parameters for IR TX or TX

Definition

```
struct v4l2_subdev_ir_parameters {
  unsigned int bytes_per_data_element;
  enum v4l2_subdev_ir_mode mode;
  bool enable;
  bool interrupt_enable;
  bool shutdown;
  bool modulation;
  u32 max_pulse_width;
  unsigned int carrier_freq;
  unsigned int duty_cycle;
  bool invert_level;
  bool invert_carrier_sense;
  u32 noise_filter_min_width;
  unsigned int carrier_range_lower;
  unsigned int carrier_range_upper;
  u32 resolution;
};
```

Members

bytes_per_data_element bytes per data element of data in read or write call.

mode IR mode as defined by *enum v4l2_subdev_ir_mode*.

enable device is active if true

interrupt_enable IR interrupts are enabled if true

shutdown if true: set hardware to low/no power, false: normal mode

modulation if true, it uses carrier, if false: baseband

max_pulse_width maximum pulse width in ns, valid only for baseband signal

carrier_freq carrier frequency in Hz, valid only for modulated signal

duty_cycle duty cycle percentage, valid only for modulated signal

invert_level invert signal level

invert_carrier_sense Send 0/space as a carrier burst. used only in TX.

noise_filter_min_width min time of a valid pulse, in ns. Used only for RX.

carrier_range_lower Lower carrier range, in Hz, valid only for modulated signal. Used only for RX.

carrier_range_upper Upper carrier range, in Hz, valid only for modulated signal. Used only for RX.

resolution The receive resolution, in ns . Used only for RX.

struct **v4l2_subdev_ir_ops**
operations for IR subdevices

Definition

```
struct v4l2_subdev_ir_ops {
  int (* rx_read) (struct v4l2_subdev *sd, u8 *buf, size_t count,ssize_t *num);
  int (* rx_g_parameters) (struct v4l2_subdev *sd,struct v4l2_subdev_ir_parameters *params);
  int (* rx_s_parameters) (struct v4l2_subdev *sd,struct v4l2_subdev_ir_parameters *params);
  int (* tx_write) (struct v4l2_subdev *sd, u8 *buf, size_t count,ssize_t *num);
  int (* tx_g_parameters) (struct v4l2_subdev *sd,struct v4l2_subdev_ir_parameters *params);
  int (* tx_s_parameters) (struct v4l2_subdev *sd,struct v4l2_subdev_ir_parameters *params);
};
```

Members

rx_read Reads received codes or pulse width data. The semantics are similar to a non-blocking `read()` call.

rx_g_parameters Get the current operating parameters and state of the the IR receiver.

rx_s_parameters Set the current operating parameters and state of the the IR receiver. It is recommended to call [rt]x_g_parameters first to fill out the current state, and only change the fields that need to be changed. Upon return, the actual device operating parameters and state will be returned. Note that hardware limitations may prevent the actual settings from matching the requested settings - e.g. an actual carrier setting of 35,904 Hz when 36,000 Hz was requested. An exception is when the shutdown parameter is true. The last used operational parameters will be returned, but the actual state of the hardware be different to minimize power consumption and processing when shutdown is true.

tx_write Writes codes or pulse width data for transmission. The semantics are similar to a non-blocking *write()* call.

tx_g_parameters Get the current operating parameters and state of the the IR transmitter.

tx_s_parameters Set the current operating parameters and state of the the IR transmitter. It is recommended to call [rt]x_g_parameters first to fill out the current state, and only change the fields that need to be changed. Upon return, the actual device operating parameters and state will be returned. Note that hardware limitations may prevent the actual settings from matching the requested settings - e.g. an actual carrier setting of 35,904 Hz when 36,000 Hz was requested. An exception is when the shutdown parameter is true. The last used operational parameters will be returned, but the actual state of the hardware be different to minimize power consumption and processing when shutdown is true.

struct **v4l2_subdev_pad_config**
Used for storing subdev pad information.

Definition

```
struct v4l2_subdev_pad_config {
  struct v4l2_mbus_framefmt try_fmt;
  struct v4l2_rect try_crop;
  struct v4l2_rect try_compose;
};
```

Members

try_fmt *struct v4l2_mbus_framefmt*

try_crop *struct v4l2_rect* to be used for crop

try_compose *struct v4l2_rect* to be used for compose

Description

This structure only needs to be passed to the pad op if the 'which' field of the main argument is set to `V4L2_SUBDEV_FORMAT_TRY`. For `V4L2_SUBDEV_FORMAT_ACTIVE` it is safe to pass `NULL`.

struct **v4l2_subdev_pad_ops**
v4l2-subdev pad level operations

Definition

```
struct v4l2_subdev_pad_ops {
  int (* init_cfg) (struct v4l2_subdev *sd,struct v4l2_subdev_pad_config *cfg);
  int (* enum_mbus_code) (struct v4l2_subdev *sd,struct v4l2_subdev_pad_config *cfg,struct v4l2_
 ↪subdev_mbus_code_enum *code);
  int (* enum_frame_size) (struct v4l2_subdev *sd,struct v4l2_subdev_pad_config *cfg,struct␣
 ↪v4l2_subdev_frame_size_enum *fse);
  int (* enum_frame_interval) (struct v4l2_subdev *sd,struct v4l2_subdev_pad_config *cfg,struct␣
 ↪v4l2_subdev_frame_interval_enum *fie);
  int (* get_fmt) (struct v4l2_subdev *sd,struct v4l2_subdev_pad_config *cfg,struct v4l2_subdev_
 ↪format *format);
  int (* set_fmt) (struct v4l2_subdev *sd,struct v4l2_subdev_pad_config *cfg,struct v4l2_subdev_
 ↪format *format);
  int (* get_selection) (struct v4l2_subdev *sd,struct v4l2_subdev_pad_config *cfg,struct v4l2_
 ↪subdev_selection *sel);
  int (* set_selection) (struct v4l2_subdev *sd,struct v4l2_subdev_pad_config *cfg,struct v4l2_
 ↪subdev_selection *sel);
  int (* get_edid) (struct v4l2_subdev *sd, struct v4l2_edid *edid);
  int (* set_edid) (struct v4l2_subdev *sd, struct v4l2_edid *edid);
  int (* dv_timings_cap) (struct v4l2_subdev *sd,struct v4l2_dv_timings_cap *cap);
  int (* enum_dv_timings) (struct v4l2_subdev *sd,struct v4l2_enum_dv_timings *timings);
#ifdef CONFIG_MEDIA_CONTROLLER
  int (* link_validate) (struct v4l2_subdev *sd, struct media_link *link,struct v4l2_subdev_
 ↪format *source_fmt,struct v4l2_subdev_format *sink_fmt);
#endif
  int (* get_frame_desc) (struct v4l2_subdev *sd, unsigned int pad,struct v4l2_mbus_frame_desc␣
 ↪*fd);
  int (* set_frame_desc) (struct v4l2_subdev *sd, unsigned int pad,struct v4l2_mbus_frame_desc␣
 ↪*fd);
};
```

Members

init_cfg initialize the pad config to default values

enum_mbus_code callback for VIDIOC_SUBDEV_ENUM_MBUS_CODE ioctl handler code.

enum_frame_size callback for VIDIOC_SUBDEV_ENUM_FRAME_SIZE ioctl handler code.

enum_frame_interval callback for VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL ioctl handler code.

get_fmt callback for VIDIOC_SUBDEV_G_FMT ioctl handler code.

set_fmt callback for VIDIOC_SUBDEV_S_FMT ioctl handler code.

get_selection callback for VIDIOC_SUBDEV_G_SELECTION ioctl handler code.

set_selection callback for VIDIOC_SUBDEV_S_SELECTION ioctl handler code.

get_edid callback for VIDIOC_SUBDEV_G_EDID ioctl handler code.

set_edid callback for VIDIOC_SUBDEV_S_EDID ioctl handler code.

dv_timings_cap callback for VIDIOC_SUBDEV_DV_TIMINGS_CAP ioctl handler code.

enum_dv_timings callback for VIDIOC_SUBDEV_ENUM_DV_TIMINGS ioctl handler code.

link_validate used by the media controller code to check if the links that belongs to a pipeline can be used for stream.

get_frame_desc get the current low level media bus frame parameters.

set_frame_desc set the low level media bus frame parameters, **fd** array may be adjusted by the subdev driver to device capabilities.

struct **v4l2_subdev_ops**
Subdev operations

Definition

```
struct v4l2_subdev_ops {
  const struct v4l2_subdev_core_ops * core;
  const struct v4l2_subdev_tuner_ops * tuner;
  const struct v4l2_subdev_audio_ops * audio;
  const struct v4l2_subdev_video_ops * video;
  const struct v4l2_subdev_vbi_ops * vbi;
  const struct v4l2_subdev_ir_ops * ir;
  const struct v4l2_subdev_sensor_ops * sensor;
  const struct v4l2_subdev_pad_ops * pad;
};
```

Members

core pointer to *struct v4l2_subdev_core_ops*. Can be NULL

tuner pointer to *struct v4l2_subdev_tuner_ops*. Can be NULL

audio pointer to *struct v4l2_subdev_audio_ops*. Can be NULL

video pointer to *struct v4l2_subdev_video_ops*. Can be NULL

vbi pointer to *struct v4l2_subdev_vbi_ops*. Can be NULL

ir pointer to *struct v4l2_subdev_ir_ops*. Can be NULL

sensor pointer to *struct v4l2_subdev_sensor_ops*. Can be NULL

pad pointer to *struct v4l2_subdev_pad_ops*. Can be NULL

struct **v4l2_subdev_internal_ops**
V4L2 subdev internal ops

Definition

```
struct v4l2_subdev_internal_ops {
  int (* registered) (struct v4l2_subdev *sd);
  void (* unregistered) (struct v4l2_subdev *sd);
  int (* open) (struct v4l2_subdev *sd, struct v4l2_subdev_fh *fh);
  int (* close) (struct v4l2_subdev *sd, struct v4l2_subdev_fh *fh);
};
```

Members

registered called when this subdev is registered. When called the v4l2_dev field is set to the correct v4l2_device.

unregistered called when this subdev is unregistered. When called the v4l2_dev field is still set to the correct v4l2_device.

open called when the subdev device node is opened by an application.

close called when the subdev device node is closed.

Description

Note:

Never call this from drivers, only the v4l2 framework can call these ops.

struct **v4l2_subdev_platform_data**
regulators config struct

Definition

```
struct v4l2_subdev_platform_data {
  struct regulator_bulk_data * regulators;
  int num_regulators;
  void * host_priv;
};
```

Members

regulators Optional regulators used to power on/off the subdevice

num_regulators Number of regululators

host_priv Per-subdevice data, specific for a certain video host device

struct **v4l2_subdev**
describes a V4L2 sub-device

Definition

```
struct v4l2_subdev {
#if defined(CONFIG_MEDIA_CONTROLLER)
  struct media_entity entity;
#endif
  struct list_head list;
  struct module * owner;
  bool owner_v4l2_dev;
  u32 flags;
  struct v4l2_device * v4l2_dev;
  const struct v4l2_subdev_ops * ops;
  const struct v4l2_subdev_internal_ops * internal_ops;
  struct v4l2_ctrl_handler * ctrl_handler;
  char name[V4L2_SUBDEV_NAME_SIZE];
  u32 grp_id;
  void * dev_priv;
  void * host_priv;
  struct video_device * devnode;
  struct device * dev;
  struct device_node * of_node;
  struct list_head async_list;
  struct v4l2_async_subdev * asd;
  struct v4l2_async_notifier * notifier;
  struct v4l2_subdev_platform_data * pdata;
};
```

Members

entity pointer to *struct media_entity*

list List of sub-devices

owner The owner is the same as the driver's `struct device` owner.

owner_v4l2_dev true if the `sd->owner` matches the owner of **v4l2_dev**->dev ownner. Initialized by *v4l2_device_register_subdev()*.

flags subdev flags. Can be: `V4L2_SUBDEV_FL_IS_I2C` - Set this flag if this subdev is a i2c device; `V4L2_SUBDEV_FL_IS_SPI` - Set this flag if this subdev is a spi device; `V4L2_SUBDEV_FL_HAS_DEVNODE` - Set this flag if this subdev needs a device node; `V4L2_SUBDEV_FL_HAS_EVENTS` - Set this flag if this subdev generates events.

v4l2_dev pointer to struct *v4l2_device*

ops pointer to struct *v4l2_subdev_ops*

internal_ops pointer to struct *v4l2_subdev_internal_ops*. Never call these internal ops from within a driver!

ctrl_handler The control handler of this subdev. May be NULL.

name[V4L2_SUBDEV_NAME_SIZE] Name of the sub-device. Please notice that the name must be unique.

grp_id can be used to group similar subdevs. Value is driver-specific

dev_priv pointer to private data

host_priv pointer to private data used by the device where the subdev is attached.

devnode subdev device node

dev pointer to the physical device, if any

of_node The device_node of the subdev, usually the same as dev->of_node.

async_list Links this subdev to a global subdev_list or **notifier**->done list.

asd Pointer to respective *struct v4l2_async_subdev*.

notifier Pointer to the managing notifier.

pdata common part of subdevice platform data

Description

Each instance of a subdev driver should create this struct, either stand-alone or embedded in a larger struct.

This structure should be initialized by *v4l2_subdev_init()* or one of its variants: *v4l2_spi_subdev_init()*, *v4l2_i2c_subdev_init()*.

struct **v4l2_subdev_fh**
Used for storing subdev information per file handle

Definition

```
struct v4l2_subdev_fh {
  struct v4l2_fh vfh;
#if defined(CONFIG_VIDEO_V4L2_SUBDEV_API)
  struct v4l2_subdev_pad_config * pad;
#endif
};
```

Members

vfh pointer to struct v4l2_fh

pad pointer to v4l2_subdev_pad_config

void **v4l2_set_subdevdata**(struct *v4l2_subdev* * *sd*, void * *p*)
Sets V4L2 dev private device data

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

void * p pointer to the private device data to be stored.

void * **v4l2_get_subdevdata**(const struct *v4l2_subdev* * *sd*)
Gets V4L2 dev private device data

Parameters

const struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

Description

Returns the pointer to the private device data to be stored.

void **v4l2_set_subdev_hostdata**(struct *v4l2_subdev* * *sd*, void * *p*)
Sets V4L2 dev private host data

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

void * p pointer to the private data to be stored.

void * **v4l2_get_subdev_hostdata**(const struct *v4l2_subdev* * *sd*)
Gets V4L2 dev private data

Parameters

const struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

Description

Returns the pointer to the private host data to be stored.

int **v4l2_subdev_link_validate_default**(struct *v4l2_subdev* * *sd*, struct *media_link* * *link*, struct *v4l2_subdev_format* * *source_fmt*, struct *v4l2_subdev_format* * *sink_fmt*)
validates a media link

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

struct media_link * link pointer to *struct media_link*

struct v4l2_subdev_format * source_fmt pointer to *struct v4l2_subdev_format*

struct v4l2_subdev_format * sink_fmt pointer to *struct v4l2_subdev_format*

Description

This function ensures that width, height and the media bus pixel code are equal on both source and sink of the link.

int **v4l2_subdev_link_validate**(struct *media_link* * *link*)
validates a media link

Parameters

struct media_link * link pointer to *struct media_link*

Description

This function calls the subdev's link_validate ops to validate if a media link is valid for streaming. It also internally calls *v4l2_subdev_link_validate_default()* to ensure that width, height and the media bus pixel code are equal on both source and sink of the link.

struct *v4l2_subdev_pad_config* * **v4l2_subdev_alloc_pad_config**(struct *v4l2_subdev* * *sd*)
Allocates memory for pad config

Parameters

struct v4l2_subdev * sd pointer to struct v4l2_subdev

void **v4l2_subdev_free_pad_config**(struct *v4l2_subdev_pad_config* * *cfg*)
Frees memory allocated by *v4l2_subdev_alloc_pad_config()*.

Parameters

struct v4l2_subdev_pad_config * cfg pointer to *struct v4l2_subdev_pad_config*

void **v4l2_subdev_init**(struct *v4l2_subdev* * *sd*, const struct *v4l2_subdev_ops* * *ops*)
initializes the sub-device struct

Parameters

struct v4l2_subdev * sd pointer to the *struct v4l2_subdev* to be initialized

const struct v4l2_subdev_ops * ops pointer to *struct v4l2_subdev_ops*.

void **v4l2_subdev_notify_event**(struct *v4l2_subdev* * *sd*, const struct *v4l2_event* * *ev*)
Delivers event notification for subdevice

Parameters

struct v4l2_subdev * sd The subdev for which to deliver the event

const struct v4l2_event * ev The event to deliver

Description

Will deliver the specified event to all userspace event listeners which are subscribed to the v42l subdev event queue as well as to the bridge driver using the notify callback. The notification type for the notify callback will be `V4L2_DEVICE_NOTIFY_EVENT`.

enum **v4l2_async_match_type**
type of asynchronous subdevice logic to be used in order to identify a match

Constants

V4L2_ASYNC_MATCH_CUSTOM Match will use the logic provided by *struct v4l2_async_subdev*.match ops

V4L2_ASYNC_MATCH_DEVNAME Match will use the device name

V4L2_ASYNC_MATCH_I2C Match will check for I2C adapter ID and address

V4L2_ASYNC_MATCH_OF Match will use OF node

Description

This enum is used by the asyncrhronous sub-device logic to define the algorithm that will be used to match an asynchronous device.

struct **v4l2_async_subdev**
sub-device descriptor, as known to a bridge

Definition

```
struct v4l2_async_subdev {
  enum v4l2_async_match_type match_type;
  union match;
  struct list_head list;
};
```

Members

match_type type of match that will be used

match union of per-bus type matching data sets

list used to link struct v4l2_async_subdev objects, waiting to be probed, to a notifier->waiting list

struct **v4l2_async_notifier**
v4l2_device notifier data

Definition

```
struct v4l2_async_notifier {
  unsigned int num_subdevs;
  struct v4l2_async_subdev ** subdevs;
  struct v4l2_device * v4l2_dev;
  struct list_head waiting;
  struct list_head done;
  struct list_head list;
  int (* bound) (struct v4l2_async_notifier *notifier,struct v4l2_subdev *subdev,struct v4l2_
 ↪async_subdev *asd);
  int (* complete) (struct v4l2_async_notifier *notifier);
  void (* unbind) (struct v4l2_async_notifier *notifier,struct v4l2_subdev *subdev,struct v4l2_
 ↪async_subdev *asd);
};
```

Members

num_subdevs number of subdevices

subdevs array of pointers to subdevice descriptors

v4l2_dev pointer to struct v4l2_device

waiting list of struct v4l2_async_subdev, waiting for their drivers

done list of struct v4l2_subdev, already probed

list member in a global list of notifiers

bound a subdevice driver has successfully probed one of subdevices

complete all subdevices have been probed successfully

unbind a subdevice is leaving

int **v4l2_async_notifier_register**(struct *v4l2_device* * *v4l2_dev*, struct *v4l2_async_notifier* * *notifier*)
registers a subdevice asynchronous notifier

Parameters

struct v4l2_device * v4l2_dev pointer to *struct v4l2_device*

struct v4l2_async_notifier * notifier pointer to *struct v4l2_async_notifier*

void **v4l2_async_notifier_unregister**(struct *v4l2_async_notifier* * *notifier*)
unregisters a subdevice asynchronous notifier

Parameters

struct v4l2_async_notifier * notifier pointer to *struct v4l2_async_notifier*

int **v4l2_async_register_subdev**(struct *v4l2_subdev* * *sd*)
registers a sub-device to the asynchronous subdevice framework

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

void **v4l2_async_unregister_subdev**(struct *v4l2_subdev* * *sd*)
unregisters a sub-device to the asynchronous subdevice framework

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

2.1.11 V4L2 events

The V4L2 events provide a generic way to pass events to user space. The driver must use *v4l2_fh* to be able to support V4L2 events.

Events are defined by a type and an optional ID. The ID may refer to a V4L2 object such as a control ID. If unused, then the ID is 0.

When the user subscribes to an event the driver will allocate a number of kevent structs for that event. So every (type, ID) event tuple will have its own set of kevent structs. This guarantees that if a driver is generating lots of events of one type in a short time, then that will not overwrite events of another type.

But if you get more events of one type than the number of kevents that were reserved, then the oldest event will be dropped and the new one added.

Furthermore, the internal struct *v4l2_subscribed_event* has `merge()` and `replace()` callbacks which drivers can set. These callbacks are called when a new event is raised and there is no more room. The `replace()` callback allows you to replace the payload of the old event with that of the new event, merging any relevant data from the old payload into the new payload that replaces it. It is called when this event

type has only one kevent struct allocated. The `merge()` callback allows you to merge the oldest event payload into that of the second-oldest event payload. It is called when there are two or more kevent structs allocated.

This way no status information is lost, just the intermediate steps leading up to that state.

A good example of these `replace`/`merge` callbacks is in v4l2-event.c: `ctrls_replace()` and `ctrls_merge()` callbacks for the control event.

Note:

these callbacks can be called from interrupt context, so they must be fast.

In order to queue events to video device, drivers should call:

v4l2_event_queue (*vdev*, *ev*)

The driver's only responsibility is to fill in the type and the data fields. The other fields will be filled in by V4L2.

Event subscription

Subscribing to an event is via:

v4l2_event_subscribe (*fh*, *sub* , elems, *ops*)

This function is used to implement *video_device->* *ioctl_ops->* `vidioc_subscribe_event`, but the driver must check first if the driver is able to produce events with specified event id, and then should call *v4l2_event_subscribe()* to subscribe the event.

The elems argument is the size of the event queue for this event. If it is 0, then the framework will fill in a default value (this depends on the event type).

The ops argument allows the driver to specify a number of callbacks:

Call-back	Description
add	called when a new listener gets added (subscribing to the same event twice will only cause this callback to get called once)
del	called when a listener stops listening
re-place	replace event 'old' with event 'new'.
merge	merge event 'old' into event 'new'.

All 4 callbacks are optional, if you don't want to specify any callbacks the ops argument itself maybe `NULL`.

Unsubscribing an event

Unsubscribing to an event is via:

v4l2_event_unsubscribe (*fh*, *sub*)

This function is used to implement *video_device->* *ioctl_ops->* `vidioc_unsubscribe_event`. A driver may call *v4l2_event_unsubscribe()* directly unless it wants to be involved in unsubscription process.

The special type `V4L2_EVENT_ALL` may be used to unsubscribe all events. The drivers may want to handle this in a special way.

Check if there's a pending event

Checking if there's a pending event is via:

v4l2_event_pending (*fh*)

This function returns the number of pending events. Useful when implementing poll.

How events work

Events are delivered to user space through the poll system call. The driver can use *v4l2_fh*->wait (a wait_queue_head_t) as the argument for `poll_wait()`.

There are standard and private events. New standard events must use the smallest available event type. The drivers must allocate their events from their own class starting from class base. Class base is V4L2_EVENT_PRIVATE_START + n * 1000 where n is the lowest available number. The first event type in the class is reserved for future use, so the first available event type is 'class base + 1'.

An example on how the V4L2 events may be used can be found in the OMAP 3 ISP driver (`drivers/media/platform/omap3isp`).

A subdev can directly send an event to the *v4l2_device* notify function with V4L2_DEVICE_NOTIFY_EVENT. This allows the bridge to map the subdev that sends the event to the video node(s) associated with the subdev that need to be informed about such an event.

V4L2 event functions and data structures

struct **v4l2_kevent**
Internal kernel event struct.

Definition

```
struct v4l2_kevent {
  struct list_head list;
  struct v4l2_subscribed_event * sev;
  struct v4l2_event event;
};
```

Members

list List node for the v4l2_fh->available list.

sev Pointer to parent v4l2_subscribed_event.

event The event itself.

struct **v4l2_subscribed_event_ops**
Subscribed event operations.

Definition

```
struct v4l2_subscribed_event_ops {
  int (* add) (struct v4l2_subscribed_event *sev, unsigned int elems);
  void (* del) (struct v4l2_subscribed_event *sev);
  void (* replace) (struct v4l2_event *old, const struct v4l2_event *new);
  void (* merge) (const struct v4l2_event *old, struct v4l2_event *new);
};
```

Members

add Optional callback, called when a new listener is added

del Optional callback, called when a listener stops listening

replace Optional callback that can replace event 'old' with event 'new'.

merge Optional callback that can merge event ‘old’ into event ‘new’.

struct **v4l2_subscribed_event**
Internal struct representing a subscribed event.

Definition

```
struct v4l2_subscribed_event {
  struct list_head list;
  u32 type;
  u32 id;
  u32 flags;
  struct v4l2_fh * fh;
  struct list_head node;
  const struct v4l2_subscribed_event_ops * ops;
  unsigned int elems;
  unsigned int first;
  unsigned int in_use;
  struct v4l2_kevent events[];
};
```

Members

list List node for the v4l2_fh->subscribed list.

type Event type.

id Associated object ID (e.g. control ID). 0 if there isn’t any.

flags Copy of v4l2_event_subscription->flags.

fh Filehandle that subscribed to this event.

node List node that hooks into the object’s event list (if there is one).

ops v4l2_subscribed_event_ops

elems The number of elements in the events array.

first The index of the events containing the oldest available event.

in_use The number of queued events.

events[] An array of **elems** events.

int **v4l2_event_dequeue**(struct *v4l2_fh* * *fh*, struct *v4l2_event* * *event*, int *nonblocking*)
Dequeue events from video device.

Parameters

struct v4l2_fh * fh pointer to struct v4l2_fh

struct v4l2_event * event pointer to struct v4l2_event

int nonblocking if not zero, waits for an event to arrive

void **v4l2_event_queue**(struct *video_device* * *vdev*, const struct *v4l2_event* * *ev*)
Queue events to video device.

Parameters

struct video_device * vdev pointer to *struct video_device*

const struct v4l2_event * ev pointer to *struct v4l2_event*

Description

The event will be queued for all *struct v4l2_fh* file handlers.

Note:

The driver's only responsibility is to fill in the type and the data fields.The other fields will be filled in by V4L2.

void **v4l2_event_queue_fh**(struct *v4l2_fh * fh*, const struct *v4l2_event * ev*)
Queue events to video device.

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

const struct v4l2_event * ev pointer to *struct v4l2_event*

Description

The event will be queued only for the specified *struct v4l2_fh* file handler.

Note:

The driver's only responsibility is to fill in the type and the data fields.The other fields will be filled in by V4L2.

int **v4l2_event_pending**(struct *v4l2_fh * fh*)
Check if an event is available

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

Description

Returns the number of pending events.

int **v4l2_event_subscribe**(struct *v4l2_fh * fh*, const struct *v4l2_event_subscription * sub*, unsigned int *elems*, const struct *v4l2_subscribed_event_ops * ops*)
Subscribes to an event

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

const struct v4l2_event_subscription * sub pointer to *struct v4l2_event_subscription*

unsigned int elems size of the events queue

const struct v4l2_subscribed_event_ops * ops pointer to *v4l2_subscribed_event_ops*

Description

Note:

*if **elems** is zero, the framework will fill in a default value, with is currently 1 element.*

int **v4l2_event_unsubscribe**(struct *v4l2_fh * fh*, const struct *v4l2_event_subscription * sub*)
Unsubscribes to an event

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

const struct v4l2_event_subscription * sub pointer to *struct v4l2_event_subscription*

void **v4l2_event_unsubscribe_all**(struct *v4l2_fh* * *fh*)
Unsubscribes to all events

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

int **v4l2_event_subdev_unsubscribe**(struct *v4l2_subdev* * *sd*, struct *v4l2_fh* * *fh*, struct *v4l2_event_subscription* * *sub*)
Subdev variant of *v4l2_event_unsubscribe()*

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

struct v4l2_fh * fh pointer to *struct v4l2_fh*

struct v4l2_event_subscription * sub pointer to *struct v4l2_event_subscription*

Description

Note:

This function should be used for the `struct v4l2_subdev_core_ops` *unsubscribe_event field.*

int **v4l2_src_change_event_subscribe**(struct *v4l2_fh* * *fh*, const struct *v4l2_event_subscription* * *sub*)
helper function that calls *v4l2_event_subscribe()* if the event is `V4L2_EVENT_SOURCE_CHANGE`.

Parameters

struct v4l2_fh * fh pointer to struct v4l2_fh

const struct v4l2_event_subscription * sub pointer to *struct v4l2_event_subscription*

int **v4l2_src_change_event_subdev_subscribe**(struct *v4l2_subdev* * *sd*, struct *v4l2_fh* * *fh*, struct *v4l2_event_subscription* * *sub*)
Variant of *v4l2_event_subscribe()*, meant to subscribe only events of the type `V4L2_EVENT_SOURCE_CHANGE`.

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

struct v4l2_fh * fh pointer to *struct v4l2_fh*

struct v4l2_event_subscription * sub pointer to *struct v4l2_event_subscription*

2.1.12 V4L2 Controls

Introduction

The V4L2 control API seems simple enough, but quickly becomes very hard to implement correctly in drivers. But much of the code needed to handle controls is actually not driver specific and can be moved to the V4L core framework.

After all, the only part that a driver developer is interested in is:

1. How do I add a control?
2. How do I set the control's value? (i.e. s_ctrl)

And occasionally:

3. How do I get the control's value? (i.e. g_volatile_ctrl)
4. How do I validate the user's proposed control value? (i.e. try_ctrl)

All the rest is something that can be done centrally.

The control framework was created in order to implement all the rules of the V4L2 specification with respect to controls in a central place. And to make life as easy as possible for the driver developer.

Note that the control framework relies on the presence of a struct v4l2_device for V4L2 drivers and struct v4l2_subdev for sub-device drivers.

Objects in the framework

There are two main objects:

The v4l2_ctrl object describes the control properties and keeps track of the control's value (both the current value and the proposed new value).

v4l2_ctrl_handler is the object that keeps track of controls. It maintains a list of v4l2_ctrl objects that it owns and another list of references to controls, possibly to controls owned by other handlers.

Basic usage for V4L2 and sub-device drivers

1. Prepare the driver:

1.1) Add the handler to your driver's top-level struct:

```
struct foo_dev {
        ...
        struct v4l2_ctrl_handler ctrl_handler;
        ...
};

struct foo_dev *foo;
```

1.2) Initialize the handler:

```
v4l2_ctrl_handler_init(&foo->ctrl_handler, nr_of_controls);
```

The second argument is a hint telling the function how many controls this handler is expected to handle. It will allocate a hashtable based on this information. It is a hint only.

1.3) Hook the control handler into the driver:

1.3.1) For V4L2 drivers do this:

```
struct foo_dev {
        ...
        struct v4l2_device v4l2_dev;
        ...
        struct v4l2_ctrl_handler ctrl_handler;
        ...
};

foo->v4l2_dev.ctrl_handler = &foo->ctrl_handler;
```

Where foo->v4l2_dev is of type struct v4l2_device.

Finally, remove all control functions from your v4l2_ioctl_ops (if any): vidioc_queryctrl, vidioc_query_ext_ctrl, vidioc_querymenu, vidioc_g_ctrl, vidioc_s_ctrl, vidioc_g_ext_ctrls, vidioc_try_ext_ctrls and vidioc_s_ext_ctrls. Those are now no longer needed.

1.3.2) For sub-device drivers do this:

```
struct foo_dev {
        ...
        struct v4l2_subdev sd;
        ...
        struct v4l2_ctrl_handler ctrl_handler;
        ...
};

foo->sd.ctrl_handler = &foo->ctrl_handler;
```

Where foo->sd is of type struct v4l2_subdev.

1.4) Clean up the handler at the end:

```
v4l2_ctrl_handler_free(&foo->ctrl_handler);
```

2. Add controls:

You add non-menu controls by calling v4l2_ctrl_new_std:

```
struct v4l2_ctrl *v4l2_ctrl_new_std(struct v4l2_ctrl_handler *hdl,
                const struct v4l2_ctrl_ops *ops,
                u32 id, s32 min, s32 max, u32 step, s32 def);
```

Menu and integer menu controls are added by calling v4l2_ctrl_new_std_menu:

```
struct v4l2_ctrl *v4l2_ctrl_new_std_menu(struct v4l2_ctrl_handler *hdl,
                const struct v4l2_ctrl_ops *ops,
                u32 id, s32 max, s32 skip_mask, s32 def);
```

Menu controls with a driver specific menu are added by calling v4l2_ctrl_new_std_menu_items:

```
struct v4l2_ctrl *v4l2_ctrl_new_std_menu_items(
                struct v4l2_ctrl_handler *hdl,
                const struct v4l2_ctrl_ops *ops, u32 id, s32 max,
                s32 skip_mask, s32 def, const char * const *qmenu);
```

Integer menu controls with a driver specific menu can be added by calling v4l2_ctrl_new_int_menu:

```
struct v4l2_ctrl *v4l2_ctrl_new_int_menu(struct v4l2_ctrl_handler *hdl,
                const struct v4l2_ctrl_ops *ops,
                u32 id, s32 max, s32 def, const s64 *qmenu_int);
```

These functions are typically called right after the v4l2_ctrl_handler_init:

```
static const s64 exp_bias_qmenu[] = {
       -2, -1, 0, 1, 2
};
static const char * const test_pattern[] = {
        "Disabled",
        "Vertical Bars",
        "Solid Black",
        "Solid White",
};

v4l2_ctrl_handler_init(&foo->ctrl_handler, nr_of_controls);
v4l2_ctrl_new_std(&foo->ctrl_handler, &foo_ctrl_ops,
                V4L2_CID_BRIGHTNESS, 0, 255, 1, 128);
v4l2_ctrl_new_std(&foo->ctrl_handler, &foo_ctrl_ops,
                V4L2_CID_CONTRAST, 0, 255, 1, 128);
v4l2_ctrl_new_std_menu(&foo->ctrl_handler, &foo_ctrl_ops,
                V4L2_CID_POWER_LINE_FREQUENCY,
                V4L2_CID_POWER_LINE_FREQUENCY_60HZ, 0,
```

```
                V4L2_CID_POWER_LINE_FREQUENCY_DISABLED);
v4l2_ctrl_new_int_menu(&foo->ctrl_handler, &foo_ctrl_ops,
                V4L2_CID_EXPOSURE_BIAS,
                ARRAY_SIZE(exp_bias_qmenu) - 1,
                ARRAY_SIZE(exp_bias_qmenu) / 2 - 1,
                exp_bias_qmenu);
v4l2_ctrl_new_std_menu_items(&foo->ctrl_handler, &foo_ctrl_ops,
                V4L2_CID_TEST_PATTERN, ARRAY_SIZE(test_pattern) - 1, 0,
                0, test_pattern);
...
if (foo->ctrl_handler.error) {
        int err = foo->ctrl_handler.error;

        v4l2_ctrl_handler_free(&foo->ctrl_handler);
        return err;
}
```

The v4l2_ctrl_new_std function returns the v4l2_ctrl pointer to the new control, but if you do not need to access the pointer outside the control ops, then there is no need to store it.

The v4l2_ctrl_new_std function will fill in most fields based on the control ID except for the min, max, step and default values. These are passed in the last four arguments. These values are driver specific while control attributes like type, name, flags are all global. The control's current value will be set to the default value.

The v4l2_ctrl_new_std_menu function is very similar but it is used for menu controls. There is no min argument since that is always 0 for menu controls, and instead of a step there is a skip_mask argument: if bit X is 1, then menu item X is skipped.

The v4l2_ctrl_new_int_menu function creates a new standard integer menu control with driver-specific items in the menu. It differs from v4l2_ctrl_new_std_menu in that it doesn't have the mask argument and takes as the last argument an array of signed 64-bit integers that form an exact menu item list.

The v4l2_ctrl_new_std_menu_items function is very similar to v4l2_ctrl_new_std_menu but takes an extra parameter qmenu, which is the driver specific menu for an otherwise standard menu control. A good example for this control is the test pattern control for capture/display/sensors devices that have the capability to generate test patterns. These test patterns are hardware specific, so the contents of the menu will vary from device to device.

Note that if something fails, the function will return NULL or an error and set ctrl_handler->error to the error code. If ctrl_handler->error was already set, then it will just return and do nothing. This is also true for v4l2_ctrl_handler_init if it cannot allocate the internal data structure.

This makes it easy to init the handler and just add all controls and only check the error code at the end. Saves a lot of repetitive error checking.

It is recommended to add controls in ascending control ID order: it will be a bit faster that way.

3. Optionally force initial control setup:

```
v4l2_ctrl_handler_setup(&foo->ctrl_handler);
```

This will call s_ctrl for all controls unconditionally. Effectively this initializes the hardware to the default control values. It is recommended that you do this as this ensures that both the internal data structures and the hardware are in sync.

4. Finally: implement the v4l2_ctrl_ops

```
static const struct v4l2_ctrl_ops foo_ctrl_ops = {
        .s_ctrl = foo_s_ctrl,
};
```

Usually all you need is s_ctrl:

```
static int foo_s_ctrl(struct v4l2_ctrl *ctrl)
{
        struct foo *state = container_of(ctrl->handler, struct foo, ctrl_handler);

        switch (ctrl->id) {
        case V4L2_CID_BRIGHTNESS:
                write_reg(0x123, ctrl->val);
                break;
        case V4L2_CID_CONTRAST:
                write_reg(0x456, ctrl->val);
                break;
        }
        return 0;
}
```

The control ops are called with the v4l2_ctrl pointer as argument. The new control value has already been validated, so all you need to do is to actually update the hardware registers.

You're done! And this is sufficient for most of the drivers we have. No need to do any validation of control values, or implement QUERYCTRL, QUERY_EXT_CTRL and QUERYMENU. And G/S_CTRL as well as G/TRY/S_EXT_CTRLS are automatically supported.

Note:

The remainder sections deal with more advanced controls topics and scenarios. In practice the basic usage as described above is sufficient for most drivers.

Inheriting Controls

When a sub-device is registered with a V4L2 driver by calling v4l2_device_register_subdev() and the ctrl_handler fields of both v4l2_subdev and v4l2_device are set, then the controls of the subdev will become automatically available in the V4L2 driver as well. If the subdev driver contains controls that already exist in the V4L2 driver, then those will be skipped (so a V4L2 driver can always override a subdev control).

What happens here is that v4l2_device_register_subdev() calls v4l2_ctrl_add_handler() adding the controls of the subdev to the controls of v4l2_device.

Accessing Control Values

The following union is used inside the control framework to access control values:

```
union v4l2_ctrl_ptr {
        s32 *p_s32;
        s64 *p_s64;
        char *p_char;
        void *p;
};
```

The v4l2_ctrl struct contains these fields that can be used to access both current and new values:

```
s32 val;
struct {
        s32 val;
} cur;


union v4l2_ctrl_ptr p_new;
union v4l2_ctrl_ptr p_cur;
```

If the control has a simple s32 type type, then:

```
&ctrl->val == ctrl->p_new.p_s32
&ctrl->cur.val == ctrl->p_cur.p_s32
```

For all other types use ctrl->p_cur.p<something>. Basically the val and cur.val fields can be considered an alias since these are used so often.

Within the control ops you can freely use these. The val and cur.val speak for themselves. The p_char pointers point to character buffers of length ctrl->maximum + 1, and are always 0-terminated.

Unless the control is marked volatile the p_cur field points to the the current cached control value. When you create a new control this value is made identical to the default value. After calling v4l2_ctrl_handler_setup() this value is passed to the hardware. It is generally a good idea to call this function.

Whenever a new value is set that new value is automatically cached. This means that most drivers do not need to implement the g_volatile_ctrl() op. The exception is for controls that return a volatile register such as a signal strength read-out that changes continuously. In that case you will need to implement g_volatile_ctrl like this:

```
static int foo_g_volatile_ctrl(struct v4l2_ctrl *ctrl)
{
        switch (ctrl->id) {
        case V4L2_CID_BRIGHTNESS:
                ctrl->val = read_reg(0x123);
                break;
        }
}
```

Note that you use the ‘new value’ union as well in g_volatile_ctrl. In general controls that need to implement g_volatile_ctrl are read-only controls. If they are not, a V4L2_EVENT_CTRL_CH_VALUE will not be generated when the control changes.

To mark a control as volatile you have to set V4L2_CTRL_FLAG_VOLATILE:

```
ctrl = v4l2_ctrl_new_std(&sd->ctrl_handler, ...);
if (ctrl)
        ctrl->flags |= V4L2_CTRL_FLAG_VOLATILE;
```

For try/s_ctrl the new values (i.e. as passed by the user) are filled in and you can modify them in try_ctrl or set them in s_ctrl. The ‘cur’ union contains the current value, which you can use (but not change!) as well.

If s_ctrl returns 0 (OK), then the control framework will copy the new final values to the ‘cur’ union.

While in g_volatile/s/try_ctrl you can access the value of all controls owned by the same handler since the handler’s lock is held. If you need to access the value of controls owned by other handlers, then you have to be very careful not to introduce deadlocks.

Outside of the control ops you have to go through to helper functions to get or set a single control value safely in your driver:

```
s32 v4l2_ctrl_g_ctrl(struct v4l2_ctrl *ctrl);
int v4l2_ctrl_s_ctrl(struct v4l2_ctrl *ctrl, s32 val);
```

These functions go through the control framework just as VIDIOC_G/S_CTRL ioctls do. Don’t use these inside the control ops g_volatile/s/try_ctrl, though, that will result in a deadlock since these helpers lock the handler as well.

You can also take the handler lock yourself:

```
mutex_lock(&state->ctrl_handler.lock);
pr_info("String value is '%s'\n", ctrl1->p_cur.p_char);
```

```
pr_info("Integer value is '%s'\n", ctrl2->cur.val);
mutex_unlock(&state->ctrl_handler.lock);
```

Menu Controls

The v4l2_ctrl struct contains this union:

```
union {
        u32 step;
        u32 menu_skip_mask;
};
```

For menu controls menu_skip_mask is used. What it does is that it allows you to easily exclude certain menu items. This is used in the VIDIOC_QUERYMENU implementation where you can return -EINVAL if a certain menu item is not present. Note that VIDIOC_QUERYCTRL always returns a step value of 1 for menu controls.

A good example is the MPEG Audio Layer II Bitrate menu control where the menu is a list of standardized possible bitrates. But in practice hardware implementations will only support a subset of those. By setting the skip mask you can tell the framework which menu items should be skipped. Setting it to 0 means that all menu items are supported.

You set this mask either through the v4l2_ctrl_config struct for a custom control, or by calling v4l2_ctrl_new_std_menu().

Custom Controls

Driver specific controls can be created using v4l2_ctrl_new_custom():

```
static const struct v4l2_ctrl_config ctrl_filter = {
        .ops = &ctrl_custom_ops,
        .id = V4L2_CID_MPEG_CX2341X_VIDEO_SPATIAL_FILTER,
        .name = "Spatial Filter",
        .type = V4L2_CTRL_TYPE_INTEGER,
        .flags = V4L2_CTRL_FLAG_SLIDER,
        .max = 15,
        .step = 1,
};

ctrl = v4l2_ctrl_new_custom(&foo->ctrl_handler, &ctrl_filter, NULL);
```

The last argument is the priv pointer which can be set to driver-specific private data.

The v4l2_ctrl_config struct also has a field to set the is_private flag.

If the name field is not set, then the framework will assume this is a standard control and will fill in the name, type and flags fields accordingly.

Active and Grabbed Controls

If you get more complex relationships between controls, then you may have to activate and deactivate controls. For example, if the Chroma AGC control is on, then the Chroma Gain control is inactive. That is, you may set it, but the value will not be used by the hardware as long as the automatic gain control is on. Typically user interfaces can disable such input fields.

You can set the ‘active’ status using v4l2_ctrl_activate(). By default all controls are active. Note that the framework does not check for this flag. It is meant purely for GUIs. The function is typically called from within s_ctrl.

The other flag is the 'grabbed' flag. A grabbed control means that you cannot change it because it is in use by some resource. Typical examples are MPEG bitrate controls that cannot be changed while capturing is in progress.

If a control is set to 'grabbed' using v4l2_ctrl_grab(), then the framework will return -EBUSY if an attempt is made to set this control. The v4l2_ctrl_grab() function is typically called from the driver when it starts or stops streaming.

Control Clusters

By default all controls are independent from the others. But in more complex scenarios you can get dependencies from one control to another. In that case you need to 'cluster' them:

```
struct foo {
        struct v4l2_ctrl_handler ctrl_handler;
#define AUDIO_CL_VOLUME (0)
#define AUDIO_CL_MUTE   (1)
        struct v4l2_ctrl *audio_cluster[2];
        ...
};

state->audio_cluster[AUDIO_CL_VOLUME] =
        v4l2_ctrl_new_std(&state->ctrl_handler, ...);
state->audio_cluster[AUDIO_CL_MUTE] =
        v4l2_ctrl_new_std(&state->ctrl_handler, ...);
v4l2_ctrl_cluster(ARRAY_SIZE(state->audio_cluster), state->audio_cluster);
```

From now on whenever one or more of the controls belonging to the same cluster is set (or 'gotten', or 'tried'), only the control ops of the first control ('volume' in this example) is called. You effectively create a new composite control. Similar to how a 'struct' works in C.

So when s_ctrl is called with V4L2_CID_AUDIO_VOLUME as argument, you should set all two controls belonging to the audio_cluster:

```
static int foo_s_ctrl(struct v4l2_ctrl *ctrl)
{
        struct foo *state = container_of(ctrl->handler, struct foo, ctrl_handler);

        switch (ctrl->id) {
        case V4L2_CID_AUDIO_VOLUME: {
                struct v4l2_ctrl *mute = ctrl->cluster[AUDIO_CL_MUTE];

                write_reg(0x123, mute->val ? 0 : ctrl->val);
                break;
        }
        case V4L2_CID_CONTRAST:
                write_reg(0x456, ctrl->val);
                break;
        }
        return 0;
}
```

In the example above the following are equivalent for the VOLUME case:

```
ctrl == ctrl->cluster[AUDIO_CL_VOLUME] == state->audio_cluster[AUDIO_CL_VOLUME]
ctrl->cluster[AUDIO_CL_MUTE] == state->audio_cluster[AUDIO_CL_MUTE]
```

In practice using cluster arrays like this becomes very tiresome. So instead the following equivalent method is used:

```
struct {
        /* audio cluster */
```

```
        struct v4l2_ctrl *volume;
        struct v4l2_ctrl *mute;
};
```

The anonymous struct is used to clearly 'cluster' these two control pointers, but it serves no other purpose. The effect is the same as creating an array with two control pointers. So you can just do:

```
state->volume = v4l2_ctrl_new_std(&state->ctrl_handler, ...);
state->mute = v4l2_ctrl_new_std(&state->ctrl_handler, ...);
v4l2_ctrl_cluster(2, &state->volume);
```

And in foo_s_ctrl you can use these pointers directly: state->mute->val.

Note that controls in a cluster may be NULL. For example, if for some reason mute was never added (because the hardware doesn't support that particular feature), then mute will be NULL. So in that case we have a cluster of 2 controls, of which only 1 is actually instantiated. The only restriction is that the first control of the cluster must always be present, since that is the 'master' control of the cluster. The master control is the one that identifies the cluster and that provides the pointer to the v4l2_ctrl_ops struct that is used for that cluster.

Obviously, all controls in the cluster array must be initialized to either a valid control or to NULL.

In rare cases you might want to know which controls of a cluster actually were set explicitly by the user. For this you can check the 'is_new' flag of each control. For example, in the case of a volume/mute cluster the 'is_new' flag of the mute control would be set if the user called VIDIOC_S_CTRL for mute only. If the user would call VIDIOC_S_EXT_CTRLS for both mute and volume controls, then the 'is_new' flag would be 1 for both controls.

The 'is_new' flag is always 1 when called from v4l2_ctrl_handler_setup().

Handling autogain/gain-type Controls with Auto Clusters

A common type of control cluster is one that handles 'auto-foo/foo'-type controls. Typical examples are autogain/gain, autoexposure/exposure, autowhitebalance/red balance/blue balance. In all cases you have one control that determines whether another control is handled automatically by the hardware, or whether it is under manual control from the user.

If the cluster is in automatic mode, then the manual controls should be marked inactive and volatile. When the volatile controls are read the g_volatile_ctrl operation should return the value that the hardware's automatic mode set up automatically.

If the cluster is put in manual mode, then the manual controls should become active again and the volatile flag is cleared (so g_volatile_ctrl is no longer called while in manual mode). In addition just before switching to manual mode the current values as determined by the auto mode are copied as the new manual values.

Finally the V4L2_CTRL_FLAG_UPDATE should be set for the auto control since changing that control affects the control flags of the manual controls.

In order to simplify this a special variation of v4l2_ctrl_cluster was introduced:

```
void v4l2_ctrl_auto_cluster(unsigned ncontrols, struct v4l2_ctrl **controls,
                            u8 manual_val, bool set_volatile);
```

The first two arguments are identical to v4l2_ctrl_cluster. The third argument tells the framework which value switches the cluster into manual mode. The last argument will optionally set V4L2_CTRL_FLAG_VOLATILE for the non-auto controls. If it is false, then the manual controls are never volatile. You would typically use that if the hardware does not give you the option to read back to values as determined by the auto mode (e.g. if autogain is on, the hardware doesn't allow you to obtain the current gain value).

The first control of the cluster is assumed to be the 'auto' control.

Using this function will ensure that you don't need to handle all the complex flag and volatile handling.

VIDIOC_LOG_STATUS Support

This ioctl allow you to dump the current status of a driver to the kernel log. The v4l2_ctrl_handler_log_status(ctrl_handler, prefix) can be used to dump the value of the controls owned by the given handler to the log. You can supply a prefix as well. If the prefix didn't end with a space, then ': ' will be added for you.

Different Handlers for Different Video Nodes

Usually the V4L2 driver has just one control handler that is global for all video nodes. But you can also specify different control handlers for different video nodes. You can do that by manually setting the ctrl_handler field of struct video_device.

That is no problem if there are no subdevs involved but if there are, then you need to block the automatic merging of subdev controls to the global control handler. You do that by simply setting the ctrl_handler field in struct v4l2_device to NULL. Now v4l2_device_register_subdev() will no longer merge subdev controls.

After each subdev was added, you will then have to call v4l2_ctrl_add_handler manually to add the subdev's control handler (sd->ctrl_handler) to the desired control handler. This control handler may be specific to the video_device or for a subset of video_device's. For example: the radio device nodes only have audio controls, while the video and vbi device nodes share the same control handler for the audio and video controls.

If you want to have one handler (e.g. for a radio device node) have a subset of another handler (e.g. for a video device node), then you should first add the controls to the first handler, add the other controls to the second handler and finally add the first handler to the second. For example:

```
v4l2_ctrl_new_std(&radio_ctrl_handler, &radio_ops, V4L2_CID_AUDIO_VOLUME, ...);
v4l2_ctrl_new_std(&radio_ctrl_handler, &radio_ops, V4L2_CID_AUDIO_MUTE, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &video_ops, V4L2_CID_BRIGHTNESS, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &video_ops, V4L2_CID_CONTRAST, ...);
v4l2_ctrl_add_handler(&video_ctrl_handler, &radio_ctrl_handler, NULL);
```

The last argument to v4l2_ctrl_add_handler() is a filter function that allows you to filter which controls will be added. Set it to NULL if you want to add all controls.

Or you can add specific controls to a handler:

```
volume = v4l2_ctrl_new_std(&video_ctrl_handler, &ops, V4L2_CID_AUDIO_VOLUME, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &ops, V4L2_CID_BRIGHTNESS, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &ops, V4L2_CID_CONTRAST, ...);
```

What you should not do is make two identical controls for two handlers. For example:

```
v4l2_ctrl_new_std(&radio_ctrl_handler, &radio_ops, V4L2_CID_AUDIO_MUTE, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &video_ops, V4L2_CID_AUDIO_MUTE, ...);
```

This would be bad since muting the radio would not change the video mute control. The rule is to have one control for each hardware 'knob' that you can twiddle.

Finding Controls

Normally you have created the controls yourself and you can store the struct v4l2_ctrl pointer into your own struct.

But sometimes you need to find a control from another handler that you do not own. For example, if you have to find a volume control from a subdev.

You can do that by calling v4l2_ctrl_find:

```
struct v4l2_ctrl *volume;

volume = v4l2_ctrl_find(sd->ctrl_handler, V4L2_CID_AUDIO_VOLUME);
```

Since v4l2_ctrl_find will lock the handler you have to be careful where you use it. For example, this is not a good idea:

```
struct v4l2_ctrl_handler ctrl_handler;

v4l2_ctrl_new_std(&ctrl_handler, &video_ops, V4L2_CID_BRIGHTNESS, ...);
v4l2_ctrl_new_std(&ctrl_handler, &video_ops, V4L2_CID_CONTRAST, ...);
```

...and in video_ops.s_ctrl:

```
case V4L2_CID_BRIGHTNESS:
        contrast = v4l2_find_ctrl(&ctrl_handler, V4L2_CID_CONTRAST);
        ...
```

When s_ctrl is called by the framework the ctrl_handler.lock is already taken, so attempting to find another control from the same handler will deadlock.

It is recommended not to use this function from inside the control ops.

Inheriting Controls

When one control handler is added to another using v4l2_ctrl_add_handler, then by default all controls from one are merged to the other. But a subdev might have low-level controls that make sense for some advanced embedded system, but not when it is used in consumer-level hardware. In that case you want to keep those low-level controls local to the subdev. You can do this by simply setting the 'is_private' flag of the control to 1:

```
static const struct v4l2_ctrl_config ctrl_private = {
        .ops = &ctrl_custom_ops,
        .id = V4L2_CID_...,
        .name = "Some Private Control",
        .type = V4L2_CTRL_TYPE_INTEGER,
        .max = 15,
        .step = 1,
        .is_private = 1,
};

ctrl = v4l2_ctrl_new_custom(&foo->ctrl_handler, &ctrl_private, NULL);
```

These controls will now be skipped when v4l2_ctrl_add_handler is called.

V4L2_CTRL_TYPE_CTRL_CLASS Controls

Controls of this type can be used by GUIs to get the name of the control class. A fully featured GUI can make a dialog with multiple tabs with each tab containing the controls belonging to a particular control class. The name of each tab can be found by querying a special control with ID <control class | 1>.

Drivers do not have to care about this. The framework will automatically add a control of this type whenever the first control belonging to a new control class is added.

Adding Notify Callbacks

Sometimes the platform or bridge driver needs to be notified when a control from a sub-device driver changes. You can set a notify callback by calling this function:

```
void v4l2_ctrl_notify(struct v4l2_ctrl *ctrl,
        void (*notify)(struct v4l2_ctrl *ctrl, void *priv), void *priv);
```

Whenever the give control changes value the notify callback will be called with a pointer to the control and the priv pointer that was passed with v4l2_ctrl_notify. Note that the control's handler lock is held when the notify function is called.

There can be only one notify function per control handler. Any attempt to set another notify function will cause a WARN_ON.

v4l2_ctrl functions and data structures

union **v4l2_ctrl_ptr**
A pointer to a control value.

Definition

```
union v4l2_ctrl_ptr {
  s32 * p_s32;
  s64 * p_s64;
  u8 * p_u8;
  u16 * p_u16;
  u32 * p_u32;
  char * p_char;
  void * p;
};
```

Members

p_s32 Pointer to a 32-bit signed value.

p_s64 Pointer to a 64-bit signed value.

p_u8 Pointer to a 8-bit unsigned value.

p_u16 Pointer to a 16-bit unsigned value.

p_u32 Pointer to a 32-bit unsigned value.

p_char Pointer to a string.

p Pointer to a compound value.

struct **v4l2_ctrl_ops**
The control operations that the driver has to provide.

Definition

```
struct v4l2_ctrl_ops {
  int (* g_volatile_ctrl) (struct v4l2_ctrl *ctrl);
  int (* try_ctrl) (struct v4l2_ctrl *ctrl);
  int (* s_ctrl) (struct v4l2_ctrl *ctrl);
};
```

Members

g_volatile_ctrl Get a new value for this control. Generally only relevant for volatile (and usually read-only) controls such as a control that returns the current signal strength which changes continuously. If not set, then the currently cached value will be returned.

try_ctrl Test whether the control's value is valid. Only relevant when the usual min/max/step checks are not sufficient.

s_ctrl Actually set the new control value. s_ctrl is compulsory. The ctrl->handler->lock is held when these ops are called, so no one else can access controls owned by that handler.

struct **v4l2_ctrl_type_ops**
The control type operations that the driver has to provide.

Definition

```
struct v4l2_ctrl_type_ops {
  bool (* equal) (const struct v4l2_ctrl *ctrl, u32 idx,union v4l2_ctrl_ptr ptr1,union v4l2_
 ↪ctrl_ptr ptr2);
  void (* init) (const struct v4l2_ctrl *ctrl, u32 idx,union v4l2_ctrl_ptr ptr);
  void (* log) (const struct v4l2_ctrl *ctrl);
  int (* validate) (const struct v4l2_ctrl *ctrl, u32 idx,union v4l2_ctrl_ptr ptr);
};
```

Members

equal return true if both values are equal.

init initialize the value.

log log the value.

validate validate the value. Return 0 on success and a negative value otherwise.

v4l2_ctrl_notify_fnc
Typedef: typedef for a notify argument with a function that should be called when a control value has changed.

Syntax

```
void v4l2_ctrl_notify_fnc (struct v4l2_ctrl * ctrl,void * priv);
```

Parameters

struct v4l2_ctrl * ctrl pointer to struct *v4l2_ctrl*

void * priv control private data

Description

This typedef definition is used as an argument to *v4l2_ctrl_notify()* and as an argument at struct *v4l2_ctrl_handler*.

struct **v4l2_ctrl**
The control structure.

Definition

```
struct v4l2_ctrl {
  struct list_head node;
  struct list_head ev_subs;
  struct v4l2_ctrl_handler * handler;
  struct v4l2_ctrl ** cluster;
  unsigned int ncontrols;
  unsigned int done:1;
  unsigned int is_new:1;
  unsigned int has_changed:1;
  unsigned int is_private:1;
  unsigned int is_auto:1;
  unsigned int is_int:1;
  unsigned int is_string:1;
  unsigned int is_ptr:1;
  unsigned int is_array:1;
  unsigned int has_volatiles:1;
  unsigned int call_notify:1;
  unsigned int manual_mode_value:8;
  const struct v4l2_ctrl_ops * ops;
  const struct v4l2_ctrl_type_ops * type_ops;
  u32 id;
```

```
  const char * name;
  enum v4l2_ctrl_type type;
  s64 minimum;
  s64 maximum;
  s64 default_value;
  u32 elems;
  u32 elem_size;
  u32 dims[V4L2_CTRL_MAX_DIMS];
  u32 nr_of_dims;
  union cur;
  union v4l2_ctrl_ptr p_new;
  union v4l2_ctrl_ptr p_cur;
};
```

Members

node The list node.

ev_subs The list of control event subscriptions.

handler The handler that owns the control.

cluster Point to start of cluster array.

ncontrols Number of controls in cluster array.

done Internal flag: set for each processed control.

is_new Set when the user specified a new value for this control. It is also set when called from *v4l2_ctrl_handler_setup()*. Drivers should never set this flag.

has_changed Set when the current value differs from the new value. Drivers should never use this flag.

is_private If set, then this control is private to its handler and it will not be added to any other handlers. Drivers can set this flag.

is_auto If set, then this control selects whether the other cluster members are in 'automatic' mode or 'manual' mode. This is used for autogain/gain type clusters. Drivers should never set this flag directly.

is_int If set, then this control has a simple integer value (i.e. it uses ctrl->val).

is_string If set, then this control has type V4L2_CTRL_TYPE_STRING.

is_ptr If set, then this control is an array and/or has type >= V4L2_CTRL_COMPOUND_TYPES and/or has type V4L2_CTRL_TYPE_STRING. In other words, *struct v4l2_ext_control* uses field p to point to the data.

is_array If set, then this control contains an N-dimensional array.

has_volatiles If set, then one or more members of the cluster are volatile. Drivers should never touch this flag.

call_notify If set, then call the handler's notify function whenever the control's value changes.

manual_mode_value If the is_auto flag is set, then this is the value of the auto control that determines if that control is in manual mode. So if the value of the auto control equals this value, then the whole cluster is in manual mode. Drivers should never set this flag directly.

ops The control ops.

type_ops The control type ops.

id The control ID.

name The control name.

type The control type.

minimum The control's minimum value.

maximum The control's maximum value.

default_value The control's default value.

elems The number of elements in the N-dimensional array.

elem_size The size in bytes of the control.

dims[V4L2_CTRL_MAX_DIMS] The size of each dimension.

nr_of_dims The number of dimensions in **dims**.

cur The control's current value.

p_new The control's new value represented via an union with provides a standard way of accessing control types through a pointer.

p_cur The control's current value represented via an union with provides a standard way of accessing control types through a pointer.

struct **v4l2_ctrl_ref**
The control reference.

Definition

```
struct v4l2_ctrl_ref {
  struct list_head node;
  struct v4l2_ctrl_ref * next;
  struct v4l2_ctrl * ctrl;
  struct v4l2_ctrl_helper * helper;
};
```

Members

node List node for the sorted list.

next Single-link list node for the hash.

ctrl The actual control information.

helper Pointer to helper struct. Used internally in `prepare_ext_ctrls` function at `v4l2-ctrl.c`.

Description

Each control handler has a list of these refs. The list_head is used to keep a sorted-by-control-ID list of all controls, while the next pointer is used to link the control in the hash's bucket.

struct **v4l2_ctrl_handler**
The control handler keeps track of all the controls: both the controls owned by the handler and those inherited from other handlers.

Definition

```
struct v4l2_ctrl_handler {
  struct mutex _lock;
  struct mutex * lock;
  struct list_head ctrls;
  struct list_head ctrl_refs;
  struct v4l2_ctrl_ref * cached;
  struct v4l2_ctrl_ref ** buckets;
  v4l2_ctrl_notify_fnc notify;
  void * notify_priv;
  u16 nr_of_buckets;
  int error;
};
```

Members

_lock Default for "lock".

lock Lock to control access to this handler and its controls. May be replaced by the user right after init.

ctrls The list of controls owned by this handler.

ctrl_refs The list of control references.

cached The last found control reference. It is common that the same control is needed multiple times, so this is a simple optimization.

buckets Buckets for the hashing. Allows for quick control lookup.

notify A notify callback that is called whenever the control changes value. Note that the handler's lock is held when the notify function is called!

notify_priv Passed as argument to the v4l2_ctrl notify callback.

nr_of_buckets Total number of buckets in the array.

error The error code of the first failed control addition.

struct **v4l2_ctrl_config**
Control configuration structure.

Definition

```
struct v4l2_ctrl_config {
  const struct v4l2_ctrl_ops * ops;
  const struct v4l2_ctrl_type_ops * type_ops;
  u32 id;
  const char * name;
  enum v4l2_ctrl_type type;
  s64 min;
  s64 max;
  u64 step;
  s64 def;
  u32 dims[V4L2_CTRL_MAX_DIMS];
  u32 elem_size;
  u32 flags;
  u64 menu_skip_mask;
  const char *const * qmenu;
  const s64 * qmenu_int;
  unsigned int is_private:1;
};
```

Members

ops The control ops.

type_ops The control type ops. Only needed for compound controls.

id The control ID.

name The control name.

type The control type.

min The control's minimum value.

max The control's maximum value.

step The control's step value for non-menu controls.

def The control's default value.

dims[V4L2_CTRL_MAX_DIMS] The size of each dimension.

elem_size The size in bytes of the control.

flags The control's flags.

menu_skip_mask The control's skip mask for menu controls. This makes it easy to skip menu items that are not valid. If bit X is set, then menu item X is skipped. Of course, this only works for menus with <= 64 menu items. There are no menus that come close to that number, so this is OK. Should we ever need more, then this will have to be extended to a bit array.

qmenu A const char * array for all menu items. Array entries that are empty strings ("") correspond to non-existing menu items (this is in addition to the menu_skip_mask above). The last entry must be NULL.

qmenu_int A const s64 integer array for all menu items of the type V4L2_CTRL_TYPE_INTEGER_MENU.

is_private If set, then this control is private to its handler and it will not be added to any other handlers.

void **v4l2_ctrl_fill**(u32 *id*, const char ** *name*, enum *v4l2_ctrl_type* * *type*, s64 * *min*, s64 * *max*, u64 * *step*, s64 * *def*, u32 * *flags*)

Fill in the control fields based on the control ID.

Parameters

u32 id ID of the control

const char ** name name of the control

enum v4l2_ctrl_type * type type of the control

s64 * min minimum value for the control

s64 * max maximum value for the control

u64 * step control step

s64 * def default value for the control

u32 * flags flags to be used on the control

Description

This works for all standard V4L2 controls. For non-standard controls it will only fill in the given arguments and **name** will be NULL.

This function will overwrite the contents of **name**, **type** and **flags**. The contents of **min**, **max**, **step** and **def** may be modified depending on the type.

Note:

Do not use in drivers! It is used internally for backwards compatibility control handling only. Once all drivers are converted to use the new control framework this function will no longer be exported.

int **v4l2_ctrl_handler_init_class**(struct *v4l2_ctrl_handler* * *hdl*, unsigned int *nr_of_controls_hint*, struct lock_class_key * *key*, const char * *name*)

Initialize the control handler.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

unsigned int nr_of_controls_hint A hint of how many controls this handler is expected to refer to. This is the total number, so including any inherited controls. It doesn't have to be precise, but if it is way off, then you either waste memory (too many buckets are allocated) or the control lookup becomes slower (not enough buckets are allocated, so there are more slow list lookups). It will always work, though.

struct lock_class_key * key Used by the lock validator if CONFIG_LOCKDEP is set.

const char * name Used by the lock validator if CONFIG_LOCKDEP is set.

Description

Attention:

Never use this call directly, always use the `v4l2_ctrl_handler_init()` *macro that hides the* ***key*** *and* ***name*** *arguments.*

Return

returns an error if the buckets could not be allocated. This error will also be stored in **hdl**->error.

v4l2_ctrl_handler_init(*hdl*, *nr_of_controls_hint*)
helper function to create a static struct `lock_class_key` and calls *v4l2_ctrl_handler_init_class()*

Parameters

hdl The control handler.

nr_of_controls_hint A hint of how many controls this handler is expected to refer to. This is the total number, so including any inherited controls. It doesn't have to be precise, but if it is way off, then you either waste memory (too many buckets are allocated) or the control lookup becomes slower (not enough buckets are allocated, so there are more slow list lookups). It will always work, though.

Description

This helper function creates a static struct `lock_class_key` and calls *v4l2_ctrl_handler_init_class()*, providing a proper name for the lock validador.

Use this helper function to initialize a control handler.

void **v4l2_ctrl_handler_free**(struct *v4l2_ctrl_handler* * *hdl*)
Free all controls owned by the handler and free the control list.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

Description

Does nothing if **hdl** == NULL.

void **v4l2_ctrl_lock**(struct *v4l2_ctrl* * *ctrl*)
Helper function to lock the handler associated with the control.

Parameters

struct v4l2_ctrl * ctrl The control to lock.

void **v4l2_ctrl_unlock**(struct *v4l2_ctrl* * *ctrl*)
Helper function to unlock the handler associated with the control.

Parameters

struct v4l2_ctrl * ctrl The control to unlock.

int **v4l2_ctrl_handler_setup**(struct *v4l2_ctrl_handler* * *hdl*)
Call the s_ctrl op for all controls belonging to the handler to initialize the hardware to the current control values.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

Description

Button controls will be skipped, as are read-only controls.

If **hdl** == NULL, then this just returns 0.

void **v4l2_ctrl_handler_log_status**(struct *v4l2_ctrl_handler* * *hdl*, const char * *prefix*)
Log all controls owned by the handler.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

const char * prefix The prefix to use when logging the control values. If the prefix does not end with a space, then ": " will be added after the prefix. If **prefix** == NULL, then no prefix will be used.

Description

For use with VIDIOC_LOG_STATUS.

Does nothing if **hdl** == NULL.

struct *v4l2_ctrl* * **v4l2_ctrl_new_custom**(struct *v4l2_ctrl_handler* * *hdl*, const struct *v4l2_ctrl_config* * *cfg*, void * *priv*)
Allocate and initialize a new custom V4L2 control.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

const struct v4l2_ctrl_config * cfg The control's configuration data.

void * priv The control's driver-specific private data.

Description

If the *v4l2_ctrl* struct could not be allocated then NULL is returned and **hdl**->error is set to the error code (if it wasn't set already).

struct *v4l2_ctrl* * **v4l2_ctrl_new_std**(struct *v4l2_ctrl_handler* * *hdl*, const struct *v4l2_ctrl_ops* * *ops*, u32 *id*, s64 *min*, s64 *max*, u64 *step*, s64 *def*)
Allocate and initialize a new standard V4L2 non-menu control.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

const struct v4l2_ctrl_ops * ops The control ops.

u32 id The control ID.

s64 min The control's minimum value.

s64 max The control's maximum value.

u64 step The control's step value

s64 def The control's default value.

Description

If the *v4l2_ctrl* struct could not be allocated, or the control ID is not known, then NULL is returned and **hdl**->error is set to the appropriate error code (if it wasn't set already).

If **id** refers to a menu control, then this function will return NULL.

Use *v4l2_ctrl_new_std_menu()* when adding menu controls.

struct *v4l2_ctrl* * **v4l2_ctrl_new_std_menu**(struct *v4l2_ctrl_handler* * *hdl*, const struct *v4l2_ctrl_ops* * *ops*, u32 *id*, u8 *max*, u64 *mask*, u8 *def*)
Allocate and initialize a new standard V4L2 menu control.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

const struct v4l2_ctrl_ops * ops The control ops.

u32 id The control ID.

u8 max The control's maximum value.

u64 mask The control's skip mask for menu controls. This makes it easy to skip menu items that are not valid. If bit X is set, then menu item X is skipped. Of course, this only works for menus with <= 64 menu items. There are no menus that come close to that number, so this is OK. Should we ever need more, then this will have to be extended to a bit array.

u8 def The control's default value.

Description

Same as *v4l2_ctrl_new_std()*, but **min** is set to 0 and the **mask** value determines which menu items are to be skipped.

If **id** refers to a non-menu control, then this function will return NULL.

struct *v4l2_ctrl* * **v4l2_ctrl_new_std_menu_items**(struct *v4l2_ctrl_handler* * *hdl*, const struct *v4l2_ctrl_ops* * *ops*, u32 *id*, u8 *max*, u64 *mask*, u8 *def*, const char *const * *qmenu*)

Create a new standard V4L2 menu control with driver specific menu.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

const struct v4l2_ctrl_ops * ops The control ops.

u32 id The control ID.

u8 max The control's maximum value.

u64 mask The control's skip mask for menu controls. This makes it easy to skip menu items that are not valid. If bit X is set, then menu item X is skipped. Of course, this only works for menus with <= 64 menu items. There are no menus that come close to that number, so this is OK. Should we ever need more, then this will have to be extended to a bit array.

u8 def The control's default value.

const char *const * qmenu The new menu.

Description

Same as *v4l2_ctrl_new_std_menu()*, but **qmenu** will be the driver specific menu of this control.

struct *v4l2_ctrl* * **v4l2_ctrl_new_int_menu**(struct *v4l2_ctrl_handler* * *hdl*, const struct *v4l2_ctrl_ops* * *ops*, u32 *id*, u8 *max*, u8 *def*, const s64 * *qmenu_int*)

Create a new standard V4L2 integer menu control.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

const struct v4l2_ctrl_ops * ops The control ops.

u32 id The control ID.

u8 max The control's maximum value.

u8 def The control's default value.

const s64 * qmenu_int The control's menu entries.

Description

Same as *v4l2_ctrl_new_std_menu()*, but **mask** is set to 0 and it additionaly takes as an argument an array of integers determining the menu items.

If **id** refers to a non-integer-menu control, then this function will return NULL.

v4l2_ctrl_filter

Typedef: Typedef to define the filter function to be used when adding a control handler.

Syntax

```
bool v4l2_ctrl_filter (const struct v4l2_ctrl * ctrl);
```

Parameters

const struct v4l2_ctrl * ctrl pointer to struct *v4l2_ctrl*.

int **v4l2_ctrl_add_handler**(struct *v4l2_ctrl_handler * hdl*, struct *v4l2_ctrl_handler * add*, *v4l2_ctrl_filter filter*)
Add all controls from handler **add** to handler **hdl**.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

struct v4l2_ctrl_handler * add The control handler whose controls you want to add to the **hdl** control handler.

v4l2_ctrl_filter filter This function will filter which controls should be added.

Description

Does nothing if either of the two handlers is a NULL pointer. If **filter** is NULL, then all controls are added. Otherwise only those controls for which **filter** returns true will be added. In case of an error **hdl**->error will be set to the error code (if it wasn't set already).

bool **v4l2_ctrl_radio_filter**(const struct *v4l2_ctrl * ctrl*)
Standard filter for radio controls.

Parameters

const struct v4l2_ctrl * ctrl The control that is filtered.

Description

This will return true for any controls that are valid for radio device nodes. Those are all of the V4L2_CID_AUDIO_* user controls and all FM transmitter class controls.

This function is to be used with *v4l2_ctrl_add_handler()*.

void **v4l2_ctrl_cluster**(unsigned int *ncontrols*, struct *v4l2_ctrl ** controls*)
Mark all controls in the cluster as belonging to that cluster.

Parameters

unsigned int ncontrols The number of controls in this cluster.

struct v4l2_ctrl ** controls The cluster control array of size **ncontrols**.

void **v4l2_ctrl_auto_cluster**(unsigned int *ncontrols*, struct *v4l2_ctrl ** controls*, u8 *manual_val*, bool *set_volatile*)
Mark all controls in the cluster as belonging to that cluster and set it up for autofoo/foo-type handling.

Parameters

unsigned int ncontrols The number of controls in this cluster.

struct v4l2_ctrl ** controls The cluster control array of size **ncontrols**. The first control must be the 'auto' control (e.g. autogain, autoexposure, etc.)

u8 manual_val The value for the first control in the cluster that equals the manual setting.

bool set_volatile If true, then all controls except the first auto control will be volatile.

Description

Use for control groups where one control selects some automatic feature and the other controls are only active whenever the automatic feature is turned off (manual mode). Typical examples: autogain vs gain, auto-whitebalance vs red and blue balance, etc.

The behavior of such controls is as follows:

When the autofoo control is set to automatic, then any manual controls are set to inactive and any reads will call g_volatile_ctrl (if the control was marked volatile).

When the autofoo control is set to manual, then any manual controls will be marked active, and any reads will just return the current value without going through g_volatile_ctrl.

In addition, this function will set the V4L2_CTRL_FLAG_UPDATE flag on the autofoo control and V4L2_CTRL_FLAG_INACTIVE on the foo control(s) if autofoo is in auto mode.

struct *v4l2_ctrl* * **v4l2_ctrl_find**(struct *v4l2_ctrl_handler* * *hdl*, u32 *id*)
Find a control with the given ID.

Parameters

struct v4l2_ctrl_handler * hdl The control handler.

u32 id The control ID to find.

Description

If **hdl** == NULL this will return NULL as well. Will lock the handler so do not use from inside *v4l2_ctrl_ops*.

void **v4l2_ctrl_activate**(struct *v4l2_ctrl* * *ctrl*, bool *active*)
Make the control active or inactive.

Parameters

struct v4l2_ctrl * ctrl The control to (de)activate.

bool active True if the control should become active.

Description

This sets or clears the V4L2_CTRL_FLAG_INACTIVE flag atomically. Does nothing if **ctrl** == NULL. This will usually be called from within the s_ctrl op. The V4L2_EVENT_CTRL event will be generated afterwards.

This function assumes that the control handler is locked.

void **v4l2_ctrl_grab**(struct *v4l2_ctrl* * *ctrl*, bool *grabbed*)
Mark the control as grabbed or not grabbed.

Parameters

struct v4l2_ctrl * ctrl The control to (de)activate.

bool grabbed True if the control should become grabbed.

Description

This sets or clears the V4L2_CTRL_FLAG_GRABBED flag atomically. Does nothing if **ctrl** == NULL. The V4L2_EVENT_CTRL event will be generated afterwards. This will usually be called when starting or stopping streaming in the driver.

This function assumes that the control handler is not locked and will take the lock itself.

int **__v4l2_ctrl_modify_range**(struct *v4l2_ctrl* * *ctrl*, s64 *min*, s64 *max*, u64 *step*, s64 *def*)
Unlocked variant of *v4l2_ctrl_modify_range()*

Parameters

struct v4l2_ctrl * ctrl The control to update.

s64 min The control's minimum value.

s64 max The control's maximum value.

u64 step The control's step value

s64 def The control's default value.

Description

Update the range of a control on the fly. This works for control types INTEGER, BOOLEAN, MENU, INTEGER MENU and BITMASK. For menu controls the **step** value is interpreted as a menu_skip_mask.

An error is returned if one of the range arguments is invalid for this control type.

This function assumes that the control handler is not locked and will take the lock itself.

int **v4l2_ctrl_modify_range**(struct *v4l2_ctrl* * *ctrl*, s64 *min*, s64 *max*, u64 *step*, s64 *def*)
Update the range of a control.

Parameters

struct v4l2_ctrl * ctrl The control to update.

s64 min The control's minimum value.

s64 max The control's maximum value.

u64 step The control's step value

s64 def The control's default value.

Description

Update the range of a control on the fly. This works for control types INTEGER, BOOLEAN, MENU, INTEGER MENU and BITMASK. For menu controls the **step** value is interpreted as a menu_skip_mask.

An error is returned if one of the range arguments is invalid for this control type.

This function assumes that the control handler is not locked and will take the lock itself.

void **v4l2_ctrl_notify**(struct *v4l2_ctrl* * *ctrl*, *v4l2_ctrl_notify_fnc notify*, void * *priv*)
Function to set a notify callback for a control.

Parameters

struct v4l2_ctrl * ctrl The control.

v4l2_ctrl_notify_fnc notify The callback function.

void * priv The callback private handle, passed as argument to the callback.

Description

This function sets a callback function for the control. If **ctrl** is NULL, then it will do nothing. If **notify** is NULL, then the notify callback will be removed.

There can be only one notify. If another already exists, then a WARN_ON will be issued and the function will do nothing.

const char * **v4l2_ctrl_get_name**(u32 *id*)
Get the name of the control

Parameters

u32 id The control ID.

Description

This function returns the name of the given control ID or NULL if it isn't a known control.

const char * const * **v4l2_ctrl_get_menu**(u32 *id*)
Get the menu string array of the control

Parameters

u32 id The control ID.

Description

This function returns the NULL-terminated menu string array name of the given control ID or NULL if it isn't a known menu control.

const s64 * **v4l2_ctrl_get_int_menu**(u32 *id*, u32 * *len*)
Get the integer menu array of the control

Parameters

u32 id The control ID.

u32 * len The size of the integer array.

Description

This function returns the integer array of the given control ID or NULL if it if it isn't a known integer menu control.

s32 **v4l2_ctrl_g_ctrl**(struct *v4l2_ctrl * ctrl*)
Helper function to get the control's value from within a driver.

Parameters

struct v4l2_ctrl * ctrl The control.

Description

This returns the control's value safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for integer type controls only.

int **__v4l2_ctrl_s_ctrl**(struct *v4l2_ctrl * ctrl*, s32 *val*)
Unlocked variant of *v4l2_ctrl_s_ctrl()*.

Parameters

struct v4l2_ctrl * ctrl The control.

s32 val TheControls name new value.

Description

This sets the control's new value safely by going through the control framework. This function assumes the control's handler is already locked, allowing it to be used from within the *v4l2_ctrl_ops* functions.

This function is for integer type controls only.

int **v4l2_ctrl_s_ctrl**(struct *v4l2_ctrl * ctrl*, s32 *val*)
Helper function to set the control's value from within a driver.

Parameters

struct v4l2_ctrl * ctrl The control.

s32 val The new value.

Description

This sets the control's new value safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for integer type controls only.

s64 **v4l2_ctrl_g_ctrl_int64**(struct *v4l2_ctrl * ctrl*)
Helper function to get a 64-bit control's value from within a driver.

Parameters

struct v4l2_ctrl * ctrl The control.

Description

This returns the control's value safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for 64-bit integer type controls only.

int **__v4l2_ctrl_s_ctrl_int64**(struct *v4l2_ctrl * ctrl*, s64 *val*)
Unlocked variant of *v4l2_ctrl_s_ctrl_int64()*.

Parameters

struct v4l2_ctrl * ctrl The control.

s64 val The new value.

Description

This sets the control's new value safely by going through the control framework. This function assumes the control's handler is already locked, allowing it to be used from within the *v4l2_ctrl_ops* functions.

This function is for 64-bit integer type controls only.

int **v4l2_ctrl_s_ctrl_int64**(struct *v4l2_ctrl* * *ctrl*, s64 *val*)
Helper function to set a 64-bit control's value from within a driver.

Parameters

struct v4l2_ctrl * ctrl The control.

s64 val The new value.

Description

This sets the control's new value safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for 64-bit integer type controls only.

int **__v4l2_ctrl_s_ctrl_string**(struct *v4l2_ctrl* * *ctrl*, const char * *s*)
Unlocked variant of *v4l2_ctrl_s_ctrl_string()*.

Parameters

struct v4l2_ctrl * ctrl The control.

const char * s The new string.

Description

This sets the control's new string safely by going through the control framework. This function assumes the control's handler is already locked, allowing it to be used from within the *v4l2_ctrl_ops* functions.

This function is for string type controls only.

int **v4l2_ctrl_s_ctrl_string**(struct *v4l2_ctrl* * *ctrl*, const char * *s*)
Helper function to set a control's string value from within a driver.

Parameters

struct v4l2_ctrl * ctrl The control.

const char * s The new string. Controls name This sets the control's new string safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

Description

This function is for string type controls only.

void **v4l2_ctrl_replace**(struct *v4l2_event* * *old*, const struct *v4l2_event* * *new*)
Function to be used as a callback to *struct v4l2_subscribed_event_ops* replace()

Parameters

struct v4l2_event * old pointer to struct *v4l2_event* with the reported event;

const struct v4l2_event * new pointer to struct *v4l2_event* with the modified event;

void **v4l2_ctrl_merge**(const struct *v4l2_event* * *old*, struct *v4l2_event* * *new*)
Function to be used as a callback to *struct v4l2_subscribed_event_ops* merge()

Parameters

const struct v4l2_event * old pointer to struct *v4l2_event* with the reported event;

struct v4l2_event * new pointer to struct *v4l2_event* with the merged event;

int **v4l2_ctrl_log_status**(struct file * *file*, void * *fh*)
helper function to implement VIDIOC_LOG_STATUS ioctl

Parameters

struct file * file pointer to struct file

void * fh unused. Kept just to be compatible to the arguments expected by *struct v4l2_ioctl_ops*.vidioc_log_status.

Description

Can be used as a vidioc_log_status function that just dumps all controls associated with the filehandle.

int **v4l2_ctrl_subscribe_event**(struct *v4l2_fh * fh*, const struct *v4l2_event_subscription * sub*)
Subscribes to an event

Parameters

struct v4l2_fh * fh pointer to struct v4l2_fh

const struct v4l2_event_subscription * sub pointer to *struct v4l2_event_subscription*

Description

Can be used as a vidioc_subscribe_event function that just subscribes control events.

unsigned int **v4l2_ctrl_poll**(struct file * *file*, struct poll_table_struct * *wait*)
function to be used as a callback to the poll() That just polls for control events.

Parameters

struct file * file pointer to struct file

struct poll_table_struct * wait pointer to struct poll_table_struct

int **v4l2_queryctrl**(struct *v4l2_ctrl_handler * hdl*, struct *v4l2_queryctrl * qc*)
Helper function to implement *VIDIOC_QUERYCTRL* ioctl

Parameters

struct v4l2_ctrl_handler * hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_queryctrl * qc pointer to *struct v4l2_queryctrl*

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_query_ext_ctrl**(struct *v4l2_ctrl_handler * hdl*, struct *v4l2_query_ext_ctrl * qc*)
Helper function to implement *VIDIOC_QUERY_EXT_CTRL* ioctl

Parameters

struct v4l2_ctrl_handler * hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_query_ext_ctrl * qc pointer to *struct v4l2_query_ext_ctrl*

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_querymenu**(struct *v4l2_ctrl_handler * hdl*, struct *v4l2_querymenu * qm*)
Helper function to implement *VIDIOC_QUERYMENU* ioctl

Parameters

struct v4l2_ctrl_handler * hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_querymenu * qm pointer to *struct v4l2_querymenu*

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_g_ctrl**(struct *v4l2_ctrl_handler* * *hdl*, struct *v4l2_control* * *ctrl*)
Helper function to implement *VIDIOC_G_CTRL* ioctl

Parameters

struct v4l2_ctrl_handler * hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_control * ctrl pointer to *struct v4l2_control*

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_s_ctrl**(struct *v4l2_fh* * *fh*, struct *v4l2_ctrl_handler* * *hdl*, struct *v4l2_control* * *ctrl*)
Helper function to implement *VIDIOC_S_CTRL* ioctl

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

struct v4l2_ctrl_handler * hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_control * ctrl pointer to *struct v4l2_control*

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_g_ext_ctrls**(struct *v4l2_ctrl_handler* * *hdl*, struct *v4l2_ext_controls* * *c*)
Helper function to implement *VIDIOC_G_EXT_CTRLS* ioctl

Parameters

struct v4l2_ctrl_handler * hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_ext_controls * c pointer to *struct v4l2_ext_controls*

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_try_ext_ctrls**(struct *v4l2_ctrl_handler* * *hdl*, struct *v4l2_ext_controls* * *c*)
Helper function to implement *VIDIOC_TRY_EXT_CTRLS* ioctl

Parameters

struct v4l2_ctrl_handler * hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_ext_controls * c pointer to *struct v4l2_ext_controls*

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_s_ext_ctrls**(struct *v4l2_fh* * *fh*, struct *v4l2_ctrl_handler* * *hdl*, struct *v4l2_ext_controls* * *c*)
Helper function to implement *VIDIOC_S_EXT_CTRLS* ioctl

Parameters

struct v4l2_fh * fh pointer to *struct v4l2_fh*

struct v4l2_ctrl_handler * hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_ext_controls * c pointer to *struct v4l2_ext_controls*

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_ctrl_subdev_subscribe_event**(struct *v4l2_subdev* * *sd*, struct *v4l2_fh* * *fh*, struct *v4l2_event_subscription* * *sub*)
Helper function to implement as a *struct v4l2_subdev_core_ops* subscribe_event function that just subscribes control events.

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

struct v4l2_fh * fh pointer to *struct v4l2_fh*

struct v4l2_event_subscription * sub pointer to *struct v4l2_event_subscription*

int **v4l2_ctrl_subdev_log_status**(struct *v4l2_subdev * sd*)
Log all controls owned by subdev's control handler.

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

2.1.13 Videobuf Framework

Author: Jonathan Corbet <corbet@lwn.net>

Current as of 2.6.33

Note:

The videobuf framework was deprecated in favor of videobuf2. Shouldn't be used on new drivers.

Introduction

The videobuf layer functions as a sort of glue layer between a V4L2 driver and user space. It handles the allocation and management of buffers for the storage of video frames. There is a set of functions which can be used to implement many of the standard POSIX I/O system calls, including read(), poll(), and, happily, mmap(). Another set of functions can be used to implement the bulk of the V4L2 ioctl() calls related to streaming I/O, including buffer allocation, queueing and dequeueing, and streaming control. Using videobuf imposes a few design decisions on the driver author, but the payback comes in the form of reduced code in the driver and a consistent implementation of the V4L2 user-space API.

Buffer types

Not all video devices use the same kind of buffers. In fact, there are (at least) three common variations:

- Buffers which are scattered in both the physical and (kernel) virtual address spaces. (Almost) all user-space buffers are like this, but it makes great sense to allocate kernel-space buffers this way as well when it is possible. Unfortunately, it is not always possible; working with this kind of buffer normally requires hardware which can do scatter/gather DMA operations.
- Buffers which are physically scattered, but which are virtually contiguous; buffers allocated with vmalloc(), in other words. These buffers are just as hard to use for DMA operations, but they can be useful in situations where DMA is not available but virtually-contiguous buffers are convenient.
- Buffers which are physically contiguous. Allocation of this kind of buffer can be unreliable on fragmented systems, but simpler DMA controllers cannot deal with anything else.

Videobuf can work with all three types of buffers, but the driver author must pick one at the outset and design the driver around that decision.

[It's worth noting that there's a fourth kind of buffer: "overlay" buffers which are located within the system's video memory. The overlay functionality is considered to be deprecated for most use, but it still shows up occasionally in system-on-chip drivers where the performance benefits merit the use of this technique. Overlay buffers can be handled as a form of scattered buffer, but there are very few implementations in the kernel and a description of this technique is currently beyond the scope of this document.]

Data structures, callbacks, and initialization

Depending on which type of buffers are being used, the driver should include one of the following files:

```
<media/videobuf-dma-sg.h>           /* Physically scattered */
<media/videobuf-vmalloc.h>          /* vmalloc() buffers    */
<media/videobuf-dma-contig.h>       /* Physically contiguous */
```

The driver's data structure describing a V4L2 device should include a struct videobuf_queue instance for the management of the buffer queue, along with a list_head for the queue of available buffers. There will also need to be an interrupt-safe spinlock which is used to protect (at least) the queue.

The next step is to write four simple callbacks to help videobuf deal with the management of buffers:

```
struct videobuf_queue_ops {
    int (*buf_setup)(struct videobuf_queue *q,
                     unsigned int *count, unsigned int *size);
    int (*buf_prepare)(struct videobuf_queue *q,
                       struct videobuf_buffer *vb,
                       enum v4l2_field field);
    void (*buf_queue)(struct videobuf_queue *q,
                      struct videobuf_buffer *vb);
    void (*buf_release)(struct videobuf_queue *q,
                        struct videobuf_buffer *vb);
};
```

buf_setup() is called early in the I/O process, when streaming is being initiated; its purpose is to tell videobuf about the I/O stream. The count parameter will be a suggested number of buffers to use; the driver should check it for rationality and adjust it if need be. As a practical rule, a minimum of two buffers are needed for proper streaming, and there is usually a maximum (which cannot exceed 32) which makes sense for each device. The size parameter should be set to the expected (maximum) size for each frame of data.

Each buffer (in the form of a struct videobuf_buffer pointer) will be passed to buf_prepare(), which should set the buffer's size, width, height, and field fields properly. If the buffer's state field is VIDEOBUF_NEEDS_INIT, the driver should pass it to:

```
int videobuf_iolock(struct videobuf_queue* q, struct videobuf_buffer *vb,
                    struct v4l2_framebuffer *fbuf);
```

Among other things, this call will usually allocate memory for the buffer. Finally, the buf_prepare() function should set the buffer's state to VIDEOBUF_PREPARED.

When a buffer is queued for I/O, it is passed to buf_queue(), which should put it onto the driver's list of available buffers and set its state to VIDEOBUF_QUEUED. Note that this function is called with the queue spinlock held; if it tries to acquire it as well things will come to a screeching halt. Yes, this is the voice of experience. Note also that videobuf may wait on the first buffer in the queue; placing other buffers in front of it could again gum up the works. So use list_add_tail() to enqueue buffers.

Finally, buf_release() is called when a buffer is no longer intended to be used. The driver should ensure that there is no I/O active on the buffer, then pass it to the appropriate free routine(s):

```
/* Scatter/gather drivers */
int videobuf_dma_unmap(struct videobuf_queue *q,
                       struct videobuf_dmabuf *dma);
int videobuf_dma_free(struct videobuf_dmabuf *dma);

/* vmalloc drivers */
void videobuf_vmalloc_free (struct videobuf_buffer *buf);

/* Contiguous drivers */
void videobuf_dma_contig_free(struct videobuf_queue *q,
                              struct videobuf_buffer *buf);
```

One way to ensure that a buffer is no longer under I/O is to pass it to:

```
int videobuf_waiton(struct videobuf_buffer *vb, int non_blocking, int intr);
```

Here, vb is the buffer, non_blocking indicates whether non-blocking I/O should be used (it should be zero in the buf_release() case), and intr controls whether an interruptible wait is used.

File operations

At this point, much of the work is done; much of the rest is slipping videobuf calls into the implementation of the other driver callbacks. The first step is in the open() function, which must initialize the videobuf queue. The function to use depends on the type of buffer used:

```
void videobuf_queue_sg_init(struct videobuf_queue *q,
                            struct videobuf_queue_ops *ops,
                            struct device *dev,
                            spinlock_t *irqlock,
                            enum v4l2_buf_type type,
                            enum v4l2_field field,
                            unsigned int msize,
                            void *priv);

void videobuf_queue_vmalloc_init(struct videobuf_queue *q,
                            struct videobuf_queue_ops *ops,
                            struct device *dev,
                            spinlock_t *irqlock,
                            enum v4l2_buf_type type,
                            enum v4l2_field field,
                            unsigned int msize,
                            void *priv);

void videobuf_queue_dma_contig_init(struct videobuf_queue *q,
                                    struct videobuf_queue_ops *ops,
                                    struct device *dev,
                                    spinlock_t *irqlock,
                                    enum v4l2_buf_type type,
                                    enum v4l2_field field,
                                    unsigned int msize,
                                    void *priv);
```

In each case, the parameters are the same: q is the queue structure for the device, ops is the set of callbacks as described above, dev is the device structure for this video device, irqlock is an interrupt-safe spinlock to protect access to the data structures, type is the buffer type used by the device (cameras will use V4L2_BUF_TYPE_VIDEO_CAPTURE, for example), field describes which field is being captured (often V4L2_FIELD_NONE for progressive devices), msize is the size of any containing structure used around struct videobuf_buffer, and priv is a private data pointer which shows up in the priv_data field of struct videobuf_queue. Note that these are void functions which, evidently, are immune to failure.

V4L2 capture drivers can be written to support either of two APIs: the read() system call and the rather more complicated streaming mechanism. As a general rule, it is necessary to support both to ensure that all applications have a chance of working with the device. Videobuf makes it easy to do that with the same code. To implement read(), the driver need only make a call to one of:

```
ssize_t videobuf_read_one(struct videobuf_queue *q,
                          char __user *data, size_t count,
                          loff_t *ppos, int nonblocking);

ssize_t videobuf_read_stream(struct videobuf_queue *q,
                             char __user *data, size_t count,
                             loff_t *ppos, int vbihack, int nonblocking);
```

Either one of these functions will read frame data into data, returning the amount actually read; the difference is that videobuf_read_one() will only read a single frame, while videobuf_read_stream() will read multiple frames if they are needed to satisfy the count requested by the application. A typical driver read() implementation will start the capture engine, call one of the above functions, then stop the engine before returning (though a smarter implementation might leave the engine running for a little while in anticipation of another read() call happening in the near future).

The poll() function can usually be implemented with a direct call to:

```
unsigned int videobuf_poll_stream(struct file *file,
                                  struct videobuf_queue *q,
                                  poll_table *wait);
```

Note that the actual wait queue eventually used will be the one associated with the first available buffer.

When streaming I/O is done to kernel-space buffers, the driver must support the mmap() system call to enable user space to access the data. In many V4L2 drivers, the often-complex mmap() implementation simplifies to a single call to:

```
int videobuf_mmap_mapper(struct videobuf_queue *q,
                         struct vm_area_struct *vma);
```

Everything else is handled by the videobuf code.

The release() function requires two separate videobuf calls:

```
void videobuf_stop(struct videobuf_queue *q);
int videobuf_mmap_free(struct videobuf_queue *q);
```

The call to videobuf_stop() terminates any I/O in progress - though it is still up to the driver to stop the capture engine. The call to videobuf_mmap_free() will ensure that all buffers have been unmapped; if so, they will all be passed to the buf_release() callback. If buffers remain mapped, videobuf_mmap_free() returns an error code instead. The purpose is clearly to cause the closing of the file descriptor to fail if buffers are still mapped, but every driver in the 2.6.32 kernel cheerfully ignores its return value.

ioctl() operations

The V4L2 API includes a very long list of driver callbacks to respond to the many ioctl() commands made available to user space. A number of these - those associated with streaming I/O - turn almost directly into videobuf calls. The relevant helper functions are:

```
int videobuf_reqbufs(struct videobuf_queue *q,
                     struct v4l2_requestbuffers *req);
int videobuf_querybuf(struct videobuf_queue *q, struct v4l2_buffer *b);
int videobuf_qbuf(struct videobuf_queue *q, struct v4l2_buffer *b);
int videobuf_dqbuf(struct videobuf_queue *q, struct v4l2_buffer *b,
                   int nonblocking);
int videobuf_streamon(struct videobuf_queue *q);
int videobuf_streamoff(struct videobuf_queue *q);
```

So, for example, a VIDIOC_REQBUFS call turns into a call to the driver's vidioc_reqbufs() callback which, in turn, usually only needs to locate the proper struct videobuf_queue pointer and pass it to videobuf_reqbufs(). These support functions can replace a great deal of buffer management boilerplate in a lot of V4L2 drivers.

The vidioc_streamon() and vidioc_streamoff() functions will be a bit more complex, of course, since they will also need to deal with starting and stopping the capture engine.

Buffer allocation

Thus far, we have talked about buffers, but have not looked at how they are allocated. The scatter/gather case is the most complex on this front. For allocation, the driver can leave buffer allocation entirely up to the videobuf layer; in this case, buffers will be allocated as anonymous user-space pages and will be very scattered indeed. If the application is using user-space buffers, no allocation is needed; the videobuf layer will take care of calling get_user_pages() and filling in the scatterlist array.

If the driver needs to do its own memory allocation, it should be done in the vidioc_reqbufs() function, *after* calling videobuf_reqbufs(). The first step is a call to:

```
struct videobuf_dmabuf *videobuf_to_dma(struct videobuf_buffer *buf);
```

The returned videobuf_dmabuf structure (defined in <media/videobuf-dma-sg.h>) includes a couple of relevant fields:

```
struct scatterlist  *sglist;
int                 sglen;
```

The driver must allocate an appropriately-sized scatterlist array and populate it with pointers to the pieces of the allocated buffer; sglen should be set to the length of the array.

Drivers using the vmalloc() method need not (and cannot) concern themselves with buffer allocation at all; videobuf will handle those details. The same is normally true of contiguous-DMA drivers as well; videobuf will allocate the buffers (with dma_alloc_coherent()) when it sees fit. That means that these drivers may be trying to do high-order allocations at any time, an operation which is not always guaranteed to work. Some drivers play tricks by allocating DMA space at system boot time; videobuf does not currently play well with those drivers.

As of 2.6.31, contiguous-DMA drivers can work with a user-supplied buffer, as long as that buffer is physically contiguous. Normal user-space allocations will not meet that criterion, but buffers obtained from other kernel drivers, or those contained within huge pages, will work with these drivers.

Filling the buffers

The final part of a videobuf implementation has no direct callback - it's the portion of the code which actually puts frame data into the buffers, usually in response to interrupts from the device. For all types of drivers, this process works approximately as follows:

- Obtain the next available buffer and make sure that somebody is actually waiting for it.
- Get a pointer to the memory and put video data there.
- Mark the buffer as done and wake up the process waiting for it.

Step (1) above is done by looking at the driver-managed list_head structure - the one which is filled in the buf_queue() callback. Because starting the engine and enqueueing buffers are done in separate steps, it's possible for the engine to be running without any buffers available - in the vmalloc() case especially. So the driver should be prepared for the list to be empty. It is equally possible that nobody is yet interested in the buffer; the driver should not remove it from the list or fill it until a process is waiting on it. That test can be done by examining the buffer's done field (a wait_queue_head_t structure) with waitqueue_active().

A buffer's state should be set to VIDEOBUF_ACTIVE before being mapped for DMA; that ensures that the videobuf layer will not try to do anything with it while the device is transferring data.

For scatter/gather drivers, the needed memory pointers will be found in the scatterlist structure described above. Drivers using the vmalloc() method can get a memory pointer with:

```
void *videobuf_to_vmalloc(struct videobuf_buffer *buf);
```

For contiguous DMA drivers, the function to use is:

```
dma_addr_t videobuf_to_dma_contig(struct videobuf_buffer *buf);
```

The contiguous DMA API goes out of its way to hide the kernel-space address of the DMA buffer from drivers.

The final step is to set the size field of the relevant videobuf_buffer structure to the actual size of the captured image, set state to VIDEOBUF_DONE, then call wake_up() on the done queue. At this point, the buffer is owned by the videobuf layer and the driver should not touch it again.

Developers who are interested in more information can go into the relevant header files; there are a few low-level functions declared there which have not been talked about here. Also worthwhile is the vivi driver (drivers/media/platform/vivi.c), which is maintained as an example of how V4L2 drivers should be written. Vivi only uses the vmalloc() API, but it's good enough to get started with. Note also that all of these calls are exported GPL-only, so they will not be available to non-GPL kernel modules.

2.1.14 V4L2 videobuf2 functions and data structures

enum **vb2_memory**
type of memory model used to make the buffers visible on userspace.

Constants

VB2_MEMORY_UNKNOWN Buffer status is unknown or it is not used yet on userspace.

VB2_MEMORY_MMAP The buffers are allocated by the Kernel and it is memory mapped via `mmap()` ioctl. This model is also used when the user is using the buffers via `read()` or *`write()`* system calls.

VB2_MEMORY_USERPTR The buffers was allocated in userspace and it is memory mapped via `mmap()` ioctl.

VB2_MEMORY_DMABUF The buffers are passed to userspace via DMA buffer.

struct **vb2_mem_ops**
memory handling/memory allocator operations

Definition

```
struct vb2_mem_ops {
  void *(* alloc) (struct device *dev, unsigned long attrs,unsigned long size,enum dma_data_
 ↪direction dma_dir,gfp_t gfp_flags);
  void (* put) (void *buf_priv);
  struct dma_buf *(* get_dmabuf) (void *buf_priv, unsigned long flags);
  void *(* get_userptr) (struct device *dev, unsigned long vaddr,unsigned long size,enum dma_
 ↪data_direction dma_dir);
  void (* put_userptr) (void *buf_priv);
  void (* prepare) (void *buf_priv);
  void (* finish) (void *buf_priv);
  void *(* attach_dmabuf) (struct device *dev,struct dma_buf *dbuf,unsigned long size,enum dma_
 ↪data_direction dma_dir);
  void (* detach_dmabuf) (void *buf_priv);
  int (* map_dmabuf) (void *buf_priv);
  void (* unmap_dmabuf) (void *buf_priv);
  void *(* vaddr) (void *buf_priv);
  void *(* cookie) (void *buf_priv);
  unsigned int        (* num_users) (void *buf_priv);
  int (* mmap) (void *buf_priv, struct vm_area_struct *vma);
};
```

Members

alloc allocate video memory and, optionally, allocator private data, return ERR_PTR() on failure or a pointer to allocator private, per-buffer data on success; the returned private structure will then be passed as **buf_priv** argument to other ops in this structure. Additional gfp_flags to use when allocating the are also passed to this operation. These flags are from the gfp_flags field of vb2_queue.

put inform the allocator that the buffer will no longer be used; usually will result in the allocator freeing the buffer (if no other users of this buffer are present); the **buf_priv** argument is the allocator private per-buffer structure previously returned from the alloc callback.

get_dmabuf acquire userspace memory for a hardware operation; used for DMABUF memory types.

get_userptr acquire userspace memory for a hardware operation; used for USERPTR memory types; vaddr is the address passed to the videobuf layer when queuing a video buffer of USERPTR type; should return an allocator private per-buffer structure associated with the buffer on success, ERR_PTR() on failure; the returned private structure will then be passed as **buf_priv** argument to other ops in this structure.

put_userptr inform the allocator that a USERPTR buffer will no longer be used.

prepare called every time the buffer is passed from userspace to the driver, useful for cache synchronisation, optional.

finish called every time the buffer is passed back from the driver to the userspace, also optional.

attach_dmabuf attach a shared struct dma_buf for a hardware operation; used for DMABUF memory types; dev is the alloc device dbuf is the shared dma_buf; returns ERR_PTR() on failure; allocator private per-buffer structure on success; this needs to be used for further accesses to the buffer.

detach_dmabuf inform the exporter of the buffer that the current DMABUF buffer is no longer used; the **buf_priv** argument is the allocator private per-buffer structure previously returned from the attach_dmabuf callback.

map_dmabuf request for access to the dmabuf from allocator; the allocator of dmabuf is informed that this driver is going to use the dmabuf.

unmap_dmabuf releases access control to the dmabuf - allocator is notified that this driver is done using the dmabuf for now.

vaddr return a kernel virtual address to a given memory buffer associated with the passed private structure or NULL if no such mapping exists.

cookie return allocator specific cookie for a given memory buffer associated with the passed private structure or NULL if not available.

num_users return the current number of users of a memory buffer; return 1 if the videobuf layer (or actually the driver using it) is the only user.

mmap setup a userspace mapping for a given memory buffer under the provided virtual memory region.

Description

Those operations are used by the videobuf2 core to implement the memory handling/memory allocators for each type of supported streaming I/O method.

Note:

1. *Required ops for USERPTR types: get_userptr, put_userptr.*
2. *Required ops for MMAP types: alloc, put, num_users, mmap.*
3. *Required ops for read/write access types: alloc, put, num_users, vaddr.*
4. *Required ops for DMABUF types: attach_dmabuf, detach_dmabuf, map_dmabuf, unmap_dmabuf.*

struct **vb2_plane**
plane information

Definition

```
struct vb2_plane {
  void * mem_priv;
  struct dma_buf * dbuf;
  unsigned int dbuf_mapped;
```

```
  unsigned int bytesused;
  unsigned int length;
  unsigned int min_length;
  union m;
  unsigned int data_offset;
};
```

Members

mem_priv private data with this plane

dbuf dma_buf - shared buffer object

dbuf_mapped flag to show whether dbuf is mapped or not

bytesused number of bytes occupied by data in the plane (payload)

length size of this plane (NOT the payload) in bytes

min_length minimum required size of this plane (NOT the payload) in bytes. **length** is always greater or equal to **min_length**.

m Union with memtype-specific data (**offset**, **userptr** or **fd**).

data_offset offset in the plane to the start of data; usually 0, unless there is a header in front of the data Should contain enough information to be able to cover all the fields of struct v4l2_plane at videodev2.h

enum **vb2_io_modes**
queue access methods

Constants

VB2_MMAP driver supports MMAP with streaming API

VB2_USERPTR driver supports USERPTR with streaming API

VB2_READ driver supports read() style access

VB2_WRITE driver supports *write()* style access

VB2_DMABUF driver supports DMABUF with streaming API

enum **vb2_buffer_state**
current video buffer state

Constants

VB2_BUF_STATE_DEQUEUED buffer under userspace control

VB2_BUF_STATE_PREPARING buffer is being prepared in videobuf

VB2_BUF_STATE_PREPARED buffer prepared in videobuf and by the driver

VB2_BUF_STATE_QUEUED buffer queued in videobuf, but not in driver

VB2_BUF_STATE_REQUEUEING re-queue a buffer to the driver

VB2_BUF_STATE_ACTIVE buffer queued in driver and possibly used in a hardware operation

VB2_BUF_STATE_DONE buffer returned from driver to videobuf, but not yet dequeued to userspace

VB2_BUF_STATE_ERROR same as above, but the operation on the buffer has ended with an error, which will be reported to the userspace when it is dequeued

struct **vb2_buffer**
represents a video buffer

Definition

```
struct vb2_buffer {
  struct vb2_queue * vb2_queue;
  unsigned int index;
  unsigned int type;
  unsigned int memory;
  unsigned int num_planes;
  struct vb2_plane planes[VB2_MAX_PLANES];
  u64 timestamp;
};
```

Members

vb2_queue the queue to which this driver belongs

index id number of the buffer

type buffer type

memory the method, in which the actual data is passed

num_planes number of planes in the buffer on an internal driver queue

planes[VB2_MAX_PLANES] private per-plane information; do not change

timestamp frame timestamp in ns

struct **vb2_ops**
driver-specific callbacks

Definition

```
struct vb2_ops {
  int (* queue_setup) (struct vb2_queue *q,unsigned int *num_buffers, unsigned int *num_planes,
 ↪unsigned int sizes[], struct device *alloc_devs[]);
  void (* wait_prepare) (struct vb2_queue *q);
  void (* wait_finish) (struct vb2_queue *q);
  int (* buf_init) (struct vb2_buffer *vb);
  int (* buf_prepare) (struct vb2_buffer *vb);
  void (* buf_finish) (struct vb2_buffer *vb);
  void (* buf_cleanup) (struct vb2_buffer *vb);
  int (* start_streaming) (struct vb2_queue *q, unsigned int count);
  void (* stop_streaming) (struct vb2_queue *q);
  void (* buf_queue) (struct vb2_buffer *vb);
};
```

Members

queue_setup called from *VIDIOC_REQBUFS()* and *VIDIOC_CREATE_BUFS()* handlers before memory allocation. It can be called twice: if the original number of requested buffers could not be allocated, then it will be called a second time with the actually allocated number of buffers to verify if that is OK. The driver should return the required number of buffers in *num_buffers, the required number of planes per buffer in *num_planes, the size of each plane should be set in the sizes[] array and optional per-plane allocator specific device in the alloc_devs[] array. When called from VIDIOC_REQBUFS,() *num_planes == 0, the driver has to use the currently configured format to determine the plane sizes and *num_buffers is the total number of buffers that are being allocated. When called from VIDIOC_CREATE_BUFS,() *num_planes != 0 and it describes the requested number of planes and sizes[] contains the requested plane sizes. If either *num_planes or the requested sizes are invalid callback must return -EINVAL. In this case *num_buffers are being allocated additionally to q->num_buffers.

wait_prepare release any locks taken while calling vb2 functions; it is called before an ioctl needs to wait for a new buffer to arrive; required to avoid a deadlock in blocking access type.

wait_finish reacquire all locks released in the previous callback; required to continue operation after sleeping while waiting for a new buffer to arrive.

buf_init called once after allocating a buffer (in MMAP case) or after acquiring a new USERPTR buffer; drivers may perform additional buffer-related initialization; initialization failure (return != 0) will prevent queue setup from completing successfully; optional.

buf_prepare called every time the buffer is queued from userspace and from the *VIDIOC_PREPARE_BUF()* ioctl; drivers may perform any initialization required before each hardware operation in this callback; drivers can access/modify the buffer here as it is still synced for the CPU; drivers that support *VIDIOC_CREATE_BUFS()* must also validate the buffer size; if an error is returned, the buffer will not be queued in driver; optional.

buf_finish called before every dequeue of the buffer back to userspace; the buffer is synced for the CPU, so drivers can access/modify the buffer contents; drivers may perform any operations required before userspace accesses the buffer; optional. The buffer state can be one of the following: DONE and ERROR occur while streaming is in progress, and the PREPARED state occurs when the queue has been canceled and all pending buffers are being returned to their default DEQUEUED state. Typically you only have to do something if the state is VB2_BUF_STATE_DONE, since in all other cases the buffer contents will be ignored anyway.

buf_cleanup called once before the buffer is freed; drivers may perform any additional cleanup; optional.

start_streaming called once to enter 'streaming' state; the driver may receive buffers with **buf_queue** callback before **start_streaming** is called; the driver gets the number of already queued buffers in count parameter; driver can return an error if hardware fails, in that case all buffers that have been already given by the **buf_queue** callback are to be returned by the driver by calling *vb2_buffer_done()* with VB2_BUF_STATE_QUEUED. If you need a minimum number of buffers before you can start streaming, then set **min_buffers_needed** in the vb2_queue structure. If that is non-zero then **start_streaming** won't be called until at least that many buffers have been queued up by userspace.

stop_streaming called when 'streaming' state must be disabled; driver should stop any DMA transactions or wait until they finish and give back all buffers it got from buf_queue callback by calling *vb2_buffer_done()* with either VB2_BUF_STATE_DONE or VB2_BUF_STATE_ERROR; may use *vb2_wait_for_all_buffers()* function

buf_queue passes buffer vb to the driver; driver may start hardware operation on this buffer; driver should give the buffer back by calling *vb2_buffer_done()* function; it is allways called after calling *VIDIOC_STREAMON()* ioctl; might be called before **start_streaming** callback if user pre-queued buffers before calling *VIDIOC_STREAMON()*.

struct **vb2_buf_ops**
driver-specific callbacks

Definition

```
struct vb2_buf_ops {
  int (* verify_planes_array) (struct vb2_buffer *vb, const void *pb);
  void (* fill_user_buffer) (struct vb2_buffer *vb, void *pb);
  int (* fill_vb2_buffer) (struct vb2_buffer *vb, const void *pb,struct vb2_plane *planes);
  void (* copy_timestamp) (struct vb2_buffer *vb, const void *pb);
};
```

Members

verify_planes_array Verify that a given user space structure contains enough planes for the buffer. This is called for each dequeued buffer.

fill_user_buffer given a vb2_buffer fill in the userspace structure. For V4L2 this is a struct v4l2_buffer.

fill_vb2_buffer given a userspace structure, fill in the vb2_buffer. If the userspace structure is invalid, then this op will return an error.

copy_timestamp copy the timestamp from a userspace structure to the vb2_buffer struct.

struct **vb2_queue**
a videobuf queue

Definition

```
struct vb2_queue {
  unsigned int type;
  unsigned int io_modes;
  struct device * dev;
  unsigned long dma_attrs;
  unsigned fileio_read_once:1;
  unsigned fileio_write_immediately:1;
  unsigned allow_zero_bytesused:1;
  unsigned quirk_poll_must_check_waiting_for_buffers:1;
  struct mutex * lock;
  void * owner;
  const struct vb2_ops * ops;
  const struct vb2_mem_ops * mem_ops;
  const struct vb2_buf_ops * buf_ops;
  void * drv_priv;
  unsigned int buf_struct_size;
  u32 timestamp_flags;
  gfp_t gfp_flags;
  u32 min_buffers_needed;
};
```

Members

type private buffer type whose content is defined by the vb2-core caller. For example, for V4L2, it should match the types defined on enum *v4l2_buf_type*

io_modes supported io methods (see vb2_io_modes enum)

dev device to use for the default allocation context if the driver doesn't fill in the **alloc_devs** array.

dma_attrs DMA attributes to use for the DMA.

fileio_read_once report EOF after reading the first buffer

fileio_write_immediately queue buffer after each *write()* call

allow_zero_bytesused allow bytesused == 0 to be passed to the driver

quirk_poll_must_check_waiting_for_buffers Return POLLERR at poll when QBUF has not been called. This is a vb1 idiom that has been adopted also by vb2.

lock pointer to a mutex that protects the vb2_queue struct. The driver can set this to a mutex to let the v4l2 core serialize the queuing ioctls. If the driver wants to handle locking itself, then this should be set to NULL. This lock is not used by the videobuf2 core API.

owner The filehandle that 'owns' the buffers, i.e. the filehandle that called reqbufs, create_buffers or started fileio. This field is not used by the videobuf2 core API, but it allows drivers to easily associate an owner filehandle with the queue.

ops driver-specific callbacks

mem_ops memory allocator specific callbacks

buf_ops callbacks to deliver buffer information between user-space and kernel-space

drv_priv driver private data

buf_struct_size size of the driver-specific buffer structure; "0" indicates the driver doesn't want to use a custom buffer structure type. for example, sizeof(struct vb2_v4l2_buffer) will be used for v4l2.

timestamp_flags Timestamp flags; V4L2_BUF_FLAG_TIMESTAMP_* and V4L2_BUF_FLAG_TSTAMP_SRC_*

gfp_flags additional gfp flags used when allocating the buffers. Typically this is 0, but it may be e.g. GFP_DMA or __GFP_DMA32 to force the buffer allocation to a specific memory zone.

min_buffers_needed the minimum number of buffers needed before **start_streaming** can be called. Used when a DMA engine cannot be started unless at least this number of buffers have been queued into the driver.

void * **vb2_plane_vaddr**(struct *vb2_buffer* * *vb*, unsigned int *plane_no*)
Return a kernel virtual address of a given plane

Parameters

struct vb2_buffer * vb vb2_buffer to which the plane in question belongs to

unsigned int plane_no plane number for which the address is to be returned

Description

This function returns a kernel virtual address of a given plane if such a mapping exist, NULL otherwise.

void * **vb2_plane_cookie**(struct *vb2_buffer* * *vb*, unsigned int *plane_no*)
Return allocator specific cookie for the given plane

Parameters

struct vb2_buffer * vb vb2_buffer to which the plane in question belongs to

unsigned int plane_no plane number for which the cookie is to be returned

Description

This function returns an allocator specific cookie for a given plane if available, NULL otherwise. The allocator should provide some simple static inline function, which would convert this cookie to the allocator specific type that can be used directly by the driver to access the buffer. This can be for example physical address, pointer to scatter list or IOMMU mapping.

void **vb2_buffer_done**(struct *vb2_buffer* * *vb*, enum *vb2_buffer_state state*)
inform videobuf that an operation on a buffer is finished

Parameters

struct vb2_buffer * vb vb2_buffer returned from the driver

enum vb2_buffer_state state either `VB2_BUF_STATE_DONE` if the operation finished successfully, `VB2_BUF_STATE_ERROR` if the operation finished with an error or `VB2_BUF_STATE_QUEUED` if the driver wants to requeue buffers. If start_streaming fails then it should return buffers with state `VB2_BUF_STATE_QUEUED` to put them back into the queue.

Description

This function should be called by the driver after a hardware operation on a buffer is finished and the buffer may be returned to userspace. The driver cannot use this buffer anymore until it is queued back to it by videobuf by the means of *vb2_ops->buf_queue* callback. Only buffers previously queued to the driver by *vb2_ops->buf_queue* can be passed to this function.

While streaming a buffer can only be returned in state DONE or ERROR. The start_streaming op can also return them in case the DMA engine cannot be started for some reason. In that case the buffers should be returned with state QUEUED.

void **vb2_discard_done**(struct *vb2_queue* * *q*)
discard all buffers marked as DONE

Parameters

struct vb2_queue * q videobuf2 queue

Description

This function is intended to be used with suspend/resume operations. It discards all ‘done’ buffers as they would be too old to be requested after resume.

Drivers must stop the hardware and synchronize with interrupt handlers and/or delayed works before calling this function to make sure no buffer will be touched by the driver and/or hardware.

int **vb2_wait_for_all_buffers**(struct *vb2_queue* * *q*)
wait until all buffers are given back to vb2

Parameters

struct vb2_queue * q videobuf2 queue

Description

This function will wait until all buffers that have been given to the driver by *vb2_ops->buf_queue* are given back to vb2 with *vb2_buffer_done()*. It doesn't call `wait_prepare()`/`wait_finish()` pair. It is intended to be called with all locks taken, for example from *vb2_ops->stop_streaming* callback.

void **vb2_core_querybuf**(struct *vb2_queue* * *q*, unsigned int *index*, void * *pb*)
query video buffer information

Parameters

struct vb2_queue * q videobuf queue

unsigned int index id number of the buffer

void * pb buffer struct passed from userspace

Description

Should be called from vidioc_querybuf ioctl handler in driver. The passed buffer should have been verified. This function fills the relevant information for the userspace.

int **vb2_core_reqbufs**(struct *vb2_queue* * *q*, enum *vb2_memory memory*, unsigned int * *count*)
Initiate streaming

Parameters

struct vb2_queue * q videobuf2 queue

enum vb2_memory memory memory type

unsigned int * count requested buffer count

Description

Should be called from vidioc_reqbufs ioctl handler of a driver.

This function:

1. verifies streaming parameters passed from the userspace,
2. sets up the queue,
3. negotiates number of buffers and planes per buffer with the driver to be used during streaming,
4. allocates internal buffer structures (struct vb2_buffer), according to the agreed parameters,
5. for MMAP memory type, allocates actual video memory, using the memory handling/allocation routines provided during queue initialization

If req->count is 0, all the memory will be freed instead. If the queue has been allocated previously (by a previous vb2_reqbufs) call and the queue is not busy, memory will be reallocated.

The return values from this function are intended to be directly returned from vidioc_reqbufs handler in driver.

int **vb2_core_create_bufs**(struct *vb2_queue* * *q*, enum *vb2_memory memory*, unsigned int * *count*, unsigned int *requested_planes*, const unsigned int *requested_sizes[]*)
Allocate buffers and any required auxiliary structs

Parameters

struct vb2_queue * q videobuf2 queue

enum vb2_memory memory memory type

unsigned int * count requested buffer count

unsigned int requested_planes number of planes requested

const unsigned int requested_sizes[] *undescribed*

Description

Should be called from *VIDIOC_CREATE_BUFS()* ioctl handler of a driver. This function:

1. verifies parameter sanity
2. calls the .:c:func:*queue_setup()* queue operation
3. performs any necessary memory allocations

Return

the return values from this function are intended to be directly returned from *VIDIOC_CREATE_BUFS()* handler in driver.

int **vb2_core_prepare_buf**(struct *vb2_queue* * *q*, unsigned int *index*, void * *pb*)
Pass ownership of a buffer from userspace to the kernel

Parameters

struct vb2_queue * q videobuf2 queue

unsigned int index id number of the buffer

void * pb buffer structure passed from userspace to vidioc_prepare_buf handler in driver

Description

Should be called from vidioc_prepare_buf ioctl handler of a driver. The passed buffer should have been verified. This function calls buf_prepare callback in the driver (if provided), in which driver-specific buffer initialization can be performed,

The return values from this function are intended to be directly returned from vidioc_prepare_buf handler in driver.

int **vb2_core_qbuf**(struct *vb2_queue* * *q*, unsigned int *index*, void * *pb*)
Queue a buffer from userspace

Parameters

struct vb2_queue * q videobuf2 queue

unsigned int index id number of the buffer

void * pb buffer structure passed from userspace to vidioc_qbuf handler in driver

Description

Should be called from vidioc_qbuf ioctl handler of a driver. The passed buffer should have been verified.

This function:

1. if necessary, calls buf_prepare callback in the driver (if provided), in which driver-specific buffer initialization can be performed,
2. if streaming is on, queues the buffer in driver by the means of *vb2_ops->buf_queue* callback for processing.

The return values from this function are intended to be directly returned from vidioc_qbuf handler in driver.

int **vb2_core_dqbuf**(struct *vb2_queue* * *q*, unsigned int * *pindex*, void * *pb*, bool *nonblocking*)
Dequeue a buffer to the userspace

Parameters

struct vb2_queue * q videobuf2 queue

unsigned int * pindex pointer to the buffer index. May be NULL

void * pb buffer structure passed from userspace to vidioc_dqbuf handler in driver

bool nonblocking if true, this call will not sleep waiting for a buffer if no buffers ready for dequeuing are present. Normally the driver would be passing (file->f_flags & O_NONBLOCK) here

Description

Should be called from vidioc_dqbuf ioctl handler of a driver. The passed buffer should have been verified.

This function:

1. calls buf_finish callback in the driver (if provided), in which driver can perform any additional operations that may be required before returning the buffer to userspace, such as cache sync,
2. the buffer struct members are filled with relevant information for the userspace.

The return values from this function are intended to be directly returned from vidioc_dqbuf handler in driver.

int **vb2_core_expbuf**(struct *vb2_queue* * *q*, int * *fd*, unsigned int *type*, unsigned int *index*, unsigned int *plane*, unsigned int *flags*)
Export a buffer as a file descriptor

Parameters

struct vb2_queue * q videobuf2 queue

int * fd file descriptor associated with DMABUF (set by driver) *

unsigned int type buffer type

unsigned int index id number of the buffer

unsigned int plane index of the plane to be exported, 0 for single plane queues

unsigned int flags flags for newly created file, currently only O_CLOEXEC is supported, refer to manual of open syscall for more details

Description

The return values from this function are intended to be directly returned from vidioc_expbuf handler in driver.

int **vb2_core_queue_init**(struct *vb2_queue* * *q*)
initialize a videobuf2 queue

Parameters

struct vb2_queue * q videobuf2 queue; this structure should be allocated in driver

Description

The vb2_queue structure should be allocated by the driver. The driver is responsible of clearing it's content and setting initial values for some required entries before calling this function. q->ops, q->mem_ops, q->type and q->io_modes are mandatory. Please refer to the struct vb2_queue description in include/media/videobuf2-core.h for more information.

void **vb2_core_queue_release**(struct *vb2_queue* * *q*)
stop streaming, release the queue and free memory

Parameters

struct vb2_queue * q videobuf2 queue

Description

This function stops streaming and performs necessary clean ups, including freeing video buffer memory. The driver is responsible for freeing the vb2_queue structure itself.

void **vb2_queue_error**(struct *vb2_queue* * *q*)
signal a fatal error on the queue

Parameters

struct vb2_queue * q videobuf2 queue

Description

Flag that a fatal unrecoverable error has occurred and wake up all processes waiting on the queue. Polling will now set POLLERR and queuing and dequeuing buffers will return -EIO.

The error flag will be cleared when cancelling the queue, either from vb2_streamoff or vb2_queue_release. Drivers should thus not call this function before starting the stream, otherwise the error flag will remain set until the queue is released when closing the device node.

int **vb2_mmap**(struct *vb2_queue* * *q*, struct vm_area_struct * *vma*)
map video buffers into application address space

Parameters

struct vb2_queue * q videobuf2 queue

struct vm_area_struct * vma vma passed to the mmap file operation handler in the driver

Description

Should be called from mmap file operation handler of a driver. This function maps one plane of one of the available video buffers to userspace. To map whole video memory allocated on reqbufs, this function has to be called once per each plane per each buffer previously allocated.

When the userspace application calls mmap, it passes to it an offset returned to it earlier by the means of vidioc_querybuf handler. That offset acts as a “cookie”, which is then used to identify the plane to be mapped. This function finds a plane with a matching offset and a mapping is performed by the means of a provided memory operation.

The return values from this function are intended to be directly returned from the mmap handler in driver.

unsigned int **vb2_core_poll**(struct *vb2_queue* * *q*, struct file * *file*, poll_table * *wait*)
implements poll userspace operation

Parameters

struct vb2_queue * q videobuf2 queue

struct file * file file argument passed to the poll file operation handler

poll_table * wait wait argument passed to the poll file operation handler

Description

This function implements poll file operation handler for a driver. For CAPTURE queues, if a buffer is ready to be dequeued, the userspace will be informed that the file descriptor of a video device is available for reading. For OUTPUT queues, if a buffer is ready to be dequeued, the file descriptor will be reported as available for writing.

The return values from this function are intended to be directly returned from poll handler in driver.

vb2_thread_fnc
Typedef: callback function for use with vb2_thread

Syntax

```
int vb2_thread_fnc (struct vb2_buffer * vb,void * priv);
```

Parameters

struct vb2_buffer * vb pointer to struct *vb2_buffer*

void * priv pointer to a private pointer

Description

This is called whenever a buffer is dequeued in the thread.

int **vb2_thread_start**(struct *vb2_queue* * *q*, *vb2_thread_fnc fnc*, void * *priv*, const char * *thread_name*)
start a thread for the given queue.

Parameters

struct vb2_queue * q videobuf queue

vb2_thread_fnc fnc callback function

void * priv priv pointer passed to the callback function

const char * thread_name the name of the thread. This will be prefixed with “vb2-”.

Description

This starts a thread that will queue and dequeue until an error occurs or **vb2_thread_stop** is called.

Attention:

This function should not be used for anything else but the videobuf2-dvb support. If you think you have another good use-case for this, then please contact the linux-media mailing list first.

int **vb2_thread_stop**(struct *vb2_queue* * *q*)
stop the thread for the given queue.

Parameters

struct vb2_queue * q videobuf queue

bool **vb2_is_streaming**(struct *vb2_queue* * *q*)
return streaming status of the queue

Parameters

struct vb2_queue * q videobuf queue

bool **vb2_fileio_is_active**(struct *vb2_queue* * *q*)
return true if fileio is active.

Parameters

struct vb2_queue * q videobuf queue

Description

This returns true if `read()` or *`write()`* is used to stream the data as opposed to stream I/O. This is almost never an important distinction, except in rare cases. One such case is that using `read()` or *`write()`* to stream a format using V4L2_FIELD_ALTERNATE is not allowed since there is no way you can pass the field information of each buffer to/from userspace. A driver that supports this field format should check for this in the queue_setup op and reject it if this function returns true.

bool **vb2_is_busy**(struct *vb2_queue* * *q*)
return busy status of the queue

Parameters

struct vb2_queue * q videobuf queue

Description

This function checks if queue has any buffers allocated.

void * **vb2_get_drv_priv**(struct *vb2_queue* * *q*)
return driver private data associated with the queue

Parameters

struct vb2_queue * q videobuf queue

void **vb2_set_plane_payload**(struct *vb2_buffer* * *vb*, unsigned int *plane_no*, unsigned long *size*)
set bytesused for the plane plane_no

Parameters

struct vb2_buffer * vb buffer for which plane payload should be set

unsigned int plane_no plane number for which payload should be set

unsigned long size payload in bytes

unsigned long **vb2_get_plane_payload**(struct *vb2_buffer* * *vb*, unsigned int *plane_no*)
get bytesused for the plane plane_no

Parameters

struct vb2_buffer * vb buffer for which plane payload should be set

unsigned int plane_no plane number for which payload should be set

unsigned long **vb2_plane_size**(struct *vb2_buffer* * *vb*, unsigned int *plane_no*)
return plane size in bytes

Parameters

struct vb2_buffer * vb buffer for which plane size should be returned

unsigned int plane_no plane number for which size should be returned

bool **vb2_start_streaming_called**(struct *vb2_queue* * *q*)
return streaming status of driver

Parameters

struct vb2_queue * q videobuf queue

void **vb2_clear_last_buffer_dequeued**(struct *vb2_queue* * *q*)
clear last buffer dequeued flag of queue

Parameters

struct vb2_queue * q videobuf queue

bool **vb2_buffer_in_use**(struct *vb2_queue* * *q*, struct *vb2_buffer* * *vb*)
return true if the buffer is in use and the queue cannot be freed (by the means of REQBUFS(0)) call

Parameters

struct vb2_queue * q videobuf queue

struct vb2_buffer * vb buffer for which plane size should be returned

int **vb2_verify_memory_type**(struct *vb2_queue* * *q*, enum *vb2_memory memory*, unsigned int *type*)
Check whether the memory type and buffer type passed to a buffer operation are compatible with the queue.

Parameters

struct vb2_queue * q videobuf queue

enum vb2_memory memory memory model, as defined by enum *vb2_memory*.

unsigned int type private buffer type whose content is defined by the vb2-core caller. For example, for V4L2, it should match the types defined on enum *v4l2_buf_type*

struct **vb2_v4l2_buffer**
video buffer information for v4l2

Definition

```
struct vb2_v4l2_buffer {
  struct vb2_buffer vb2_buf;
  __u32 flags;
  __u32 field;
  struct v4l2_timecode timecode;
  __u32 sequence;
};
```

Members

vb2_buf video buffer 2

flags buffer informational flags

field enum v4l2_field; field order of the image in the buffer

timecode frame timecode

sequence sequence count of this frame

Description

Should contain enough information to be able to cover all the fields of struct v4l2_buffer at videodev2.h

int **vb2_reqbufs**(struct *vb2_queue* * *q*, struct *v4l2_requestbuffers* * *req*)
Wrapper for *vb2_core_reqbufs()* that also verifies the memory and type values.

Parameters

struct vb2_queue * q videobuf2 queue

struct v4l2_requestbuffers * req struct passed from userspace to vidioc_reqbufs handler in driver

int **vb2_create_bufs**(struct *vb2_queue* * *q*, struct *v4l2_create_buffers* * *create*)
Wrapper for *vb2_core_create_bufs()* that also verifies the memory and type values.

Parameters

struct vb2_queue * q videobuf2 queue

struct v4l2_create_buffers * create creation parameters, passed from userspace to vidioc_create_bufs handler in driver

int **vb2_prepare_buf**(struct *vb2_queue* * *q*, struct *v4l2_buffer* * *b*)
Pass ownership of a buffer from userspace to the kernel

Parameters

struct vb2_queue * q videobuf2 queue

struct v4l2_buffer * b buffer structure passed from userspace to vidioc_prepare_buf handler in driver

Description

Should be called from vidioc_prepare_buf ioctl handler of a driver. This function:

1. verifies the passed buffer,
2. calls buf_prepare callback in the driver (if provided), in which driver-specific buffer initialization can be performed.

The return values from this function are intended to be directly returned from vidioc_prepare_buf handler in driver.

int **vb2_qbuf**(struct *vb2_queue* * *q*, struct *v4l2_buffer* * *b*)
Queue a buffer from userspace

Parameters

struct vb2_queue * q videobuf2 queue

struct v4l2_buffer * b buffer structure passed from userspace to *VIDIOC_QBUF()* handler in driver

Description

Should be called from *VIDIOC_QBUF()* ioctl handler of a driver.

This function:

1. verifies the passed buffer,
2. if necessary, calls buf_prepare callback in the driver (if provided), in which driver-specific buffer initialization can be performed,
3. if streaming is on, queues the buffer in driver by the means of buf_queue callback for processing.

The return values from this function are intended to be directly returned from *VIDIOC_QBUF()* handler in driver.

int **vb2_expbuf**(struct *vb2_queue* * *q*, struct *v4l2_exportbuffer* * *eb*)
Export a buffer as a file descriptor

Parameters

struct vb2_queue * q videobuf2 queue

struct v4l2_exportbuffer * eb export buffer structure passed from userspace to *VIDIOC_EXPBUF()* handler in driver

Description

The return values from this function are intended to be directly returned from *VIDIOC_EXPBUF()* handler in driver.

int **vb2_dqbuf**(struct *vb2_queue* * *q*, struct *v4l2_buffer* * *b*, bool *nonblocking*)
Dequeue a buffer to the userspace

Parameters

struct vb2_queue * q videobuf2 queue

struct v4l2_buffer * b buffer structure passed from userspace to *VIDIOC_DQBUF()* handler in driver

bool nonblocking if true, this call will not sleep waiting for a buffer if no buffers ready for dequeuing are present. Normally the driver would be passing (file->f_flags & O_NONBLOCK) here

Description

Should be called from *VIDIOC_DQBUF()* ioctl handler of a driver.

This function:

1. verifies the passed buffer,
2. calls buf_finish callback in the driver (if provided), in which driver can perform any additional operations that may be required before returning the buffer to userspace, such as cache sync,
3. the buffer struct members are filled with relevant information for the userspace.

The return values from this function are intended to be directly returned from *VIDIOC_DQBUF()* handler in driver.

int **vb2_streamon**(struct *vb2_queue* * *q*, enum *v4l2_buf_type type*)
start streaming

Parameters

struct vb2_queue * q videobuf2 queue

enum v4l2_buf_type type type argument passed from userspace to vidioc_streamon handler

Description

Should be called from vidioc_streamon handler of a driver.

This function:

1. verifies current state
2. passes any previously queued buffers to the driver and starts streaming

The return values from this function are intended to be directly returned from vidioc_streamon handler in the driver.

int **vb2_streamoff**(struct *vb2_queue* * *q*, enum *v4l2_buf_type type*)
stop streaming

Parameters

struct vb2_queue * q videobuf2 queue

enum v4l2_buf_type type type argument passed from userspace to vidioc_streamoff handler

Description

Should be called from vidioc_streamoff handler of a driver.

This function:

1. verifies current state,
2. stop streaming and dequeues any queued buffers, including those previously passed to the driver (after waiting for the driver to finish).

This call can be used for pausing playback. The return values from this function are intended to be directly returned from vidioc_streamoff handler in the driver

int **vb2_queue_init**(struct *vb2_queue* * *q*)
initialize a videobuf2 queue

Parameters

struct vb2_queue * q videobuf2 queue; this structure should be allocated in driver

Description

The vb2_queue structure should be allocated by the driver. The driver is responsible of clearing it's content and setting initial values for some required entries before calling this function. q->ops, q->mem_ops, q->type and q->io_modes are mandatory. Please refer to the struct vb2_queue description in include/media/videobuf2-core.h for more information.

void **vb2_queue_release**(struct *vb2_queue* * *q*)
stop streaming, release the queue and free memory

Parameters

struct vb2_queue * q videobuf2 queue

Description

This function stops streaming and performs necessary clean ups, including freeing video buffer memory. The driver is responsible for freeing the vb2_queue structure itself.

unsigned int **vb2_poll**(struct *vb2_queue* * *q*, struct file * *file*, poll_table * *wait*)
implements poll userspace operation

Parameters

struct vb2_queue * q videobuf2 queue

struct file * file file argument passed to the poll file operation handler

poll_table * wait wait argument passed to the poll file operation handler

Description

This function implements poll file operation handler for a driver. For CAPTURE queues, if a buffer is ready to be dequeued, the userspace will be informed that the file descriptor of a video device is available for reading. For OUTPUT queues, if a buffer is ready to be dequeued, the file descriptor will be reported as available for writing.

If the driver uses struct v4l2_fh, then *vb2_poll()* will also check for any pending events.

The return values from this function are intended to be directly returned from poll handler in driver.

void **vb2_ops_wait_prepare**(struct *vb2_queue* * *vq*)
helper function to lock a struct *vb2_queue*

Parameters

struct vb2_queue * vq pointer to struct vb2_queue

Description

..note:: only use if vq->lock is non-NULL.

void **vb2_ops_wait_finish**(struct *vb2_queue* * *vq*)
helper function to unlock a struct *vb2_queue*

Parameters

struct vb2_queue * vq pointer to struct vb2_queue

Description

..note:: only use if vq->lock is non-NULL.

struct **vb2_vmarea_handler**
common vma refcount tracking handler

Definition

```
struct vb2_vmarea_handler {
  atomic_t * refcount;
  void (* put) (void *arg);
  void * arg;
};
```

Members

refcount pointer to refcount entry in the buffer

put callback to function that decreases buffer refcount

arg argument for **put** callback

2.1.15 V4L2 clocks

Attention:

This is a temporary API and it shall be replaced by the generic clock API, when the latter becomes widely available.

Many subdevices, like camera sensors, TV decoders and encoders, need a clock signal to be supplied by the system. Often this clock is supplied by the respective bridge device. The Linux kernel provides a Common Clock Framework for this purpose. However, it is not (yet) available on all architectures. Besides, the nature of the multi-functional (clock, data + synchronisation, I2C control) connection of subdevices to the system might impose special requirements on the clock API usage. E.g. V4L2 has to support clock provider driver unregistration while a subdevice driver is holding a reference to the clock. For these reasons a V4L2 clock helper API has been developed and is provided to bridge and subdevice drivers.

The API consists of two parts: two functions to register and unregister a V4L2 clock source: v4l2_clk_register() and v4l2_clk_unregister() and calls to control a clock object, similar to the respective generic clock API calls: v4l2_clk_get(), v4l2_clk_put(), v4l2_clk_enable(), v4l2_clk_disable(), v4l2_clk_get_rate(), and v4l2_clk_set_rate(). Clock suppliers have to provide clock operations that will be called when clock users invoke respective API methods.

It is expected that once the CCF becomes available on all relevant architectures this API will be removed.

2.1.16 V4L2 DV Timings functions

v4l2_check_dv_timings_fnc
Typedef: timings check callback

Syntax

```
bool v4l2_check_dv_timings_fnc (const struct v4l2_dv_timings * t,void * han-
dle);
```

Parameters

const struct v4l2_dv_timings * t the v4l2_dv_timings struct.

void * handle a handle from the driver.

Description

Returns true if the given timings are valid.

bool **v4l2_valid_dv_timings**(const struct *v4l2_dv_timings * t*, const struct *v4l2_dv_timings_cap * cap*, *v4l2_check_dv_timings_fnc fnc*, void * *fnc_handle*)
are these timings valid?

Parameters

const struct v4l2_dv_timings * t the v4l2_dv_timings struct.

const struct v4l2_dv_timings_cap * cap the v4l2_dv_timings_cap capabilities.

v4l2_check_dv_timings_fnc fnc callback to check if this timing is OK. May be NULL.

void * fnc_handle a handle that is passed on to **fnc**.

Description

Returns true if the given dv_timings struct is supported by the hardware capabilities and the callback function (if non-NULL), returns false otherwise.

int **v4l2_enum_dv_timings_cap**(struct *v4l2_enum_dv_timings * t*, const struct *v4l2_dv_timings_cap * cap*, *v4l2_check_dv_timings_fnc fnc*, void * *fnc_handle*)
Helper function to enumerate possible DV timings based on capabilities

Parameters

struct v4l2_enum_dv_timings * t the v4l2_enum_dv_timings struct.

const struct v4l2_dv_timings_cap * cap the v4l2_dv_timings_cap capabilities.

v4l2_check_dv_timings_fnc fnc callback to check if this timing is OK. May be NULL.

void * fnc_handle a handle that is passed on to **fnc**.

Description

This enumerates dv_timings using the full list of possible CEA-861 and DMT timings, filtering out any timings that are not supported based on the hardware capabilities and the callback function (if non-NULL).

If a valid timing for the given index is found, it will fill in **t** and return 0, otherwise it returns -EINVAL.

bool **v4l2_find_dv_timings_cap**(struct *v4l2_dv_timings * t*, const struct *v4l2_dv_timings_cap * cap*, unsigned *pclock_delta*, *v4l2_check_dv_timings_fnc fnc*, void * *fnc_handle*)
Find the closest timings struct

Parameters

struct v4l2_dv_timings * t the v4l2_enum_dv_timings struct.

const struct v4l2_dv_timings_cap * cap the v4l2_dv_timings_cap capabilities.

unsigned pclock_delta maximum delta between t->pixelclock and the timing struct under consideration.

v4l2_check_dv_timings_fnc fnc callback to check if a given timings struct is OK. May be NULL.

void * fnc_handle a handle that is passed on to **fnc**.

Description

This function tries to map the given timings to an entry in the full list of possible CEA-861 and DMT timings, filtering out any timings that are not supported based on the hardware capabilities and the callback function (if non-NULL).

On success it will fill in **t** with the found timings and it returns true. On failure it will return false.

bool **v4l2_find_dv_timings_cea861_vic**(struct *v4l2_dv_timings* * *t*, u8 *vic*)
find timings based on CEA-861 VIC

Parameters

struct v4l2_dv_timings * t the timings data.

u8 vic CEA-861 VIC code

Description

On success it will fill in **t** with the found timings and it returns true. On failure it will return false.

bool **v4l2_match_dv_timings**(const struct *v4l2_dv_timings* * *measured*, const struct *v4l2_dv_timings* * *standard*, unsigned *pclock_delta*, bool *match_reduced_fps*)
do two timings match?

Parameters

const struct v4l2_dv_timings * measured the measured timings data.

const struct v4l2_dv_timings * standard the timings according to the standard.

unsigned pclock_delta maximum delta in Hz between standard->pixelclock and the measured timings.

bool match_reduced_fps if true, then fail if V4L2_DV_FL_REDUCED_FPS does not match.

Description

Returns true if the two timings match, returns false otherwise.

void **v4l2_print_dv_timings**(const char * *dev_prefix*, const char * *prefix*, const struct *v4l2_dv_timings* * *t*, bool *detailed*)
log the contents of a dv_timings struct

Parameters

const char * dev_prefix device prefix for each log line.

const char * prefix additional prefix for each log line, may be NULL.

const struct v4l2_dv_timings * t the timings data.

bool detailed if true, give a detailed log.

bool **v4l2_detect_cvt**(unsigned *frame_height*, unsigned *hfreq*, unsigned *vsync*, unsigned *active_width*, u32 *polarities*, bool *interlaced*, struct *v4l2_dv_timings* * *fmt*)
detect if the given timings follow the CVT standard

Parameters

unsigned frame_height the total height of the frame (including blanking) in lines.

unsigned hfreq the horizontal frequency in Hz.

unsigned vsync the height of the vertical sync in lines.

unsigned active_width active width of image (does not include blanking). This information is needed only in case of version 2 of reduced blanking. In other cases, this parameter does not have any effect on timings.

u32 polarities the horizontal and vertical polarities (same as struct v4l2_bt_timings polarities).

bool interlaced if this flag is true, it indicates interlaced format

struct v4l2_dv_timings * fmt the resulting timings.

Description

This function will attempt to detect if the given values correspond to a valid CVT format. If so, then it will return true, and fmt will be filled in with the found CVT timings.

bool **v4l2_detect_gtf**(unsigned *frame_height*, unsigned *hfreq*, unsigned *vsync*, u32 *polarities*, bool *interlaced*, struct *v4l2_fract aspect*, struct *v4l2_dv_timings* * *fmt*)
detect if the given timings follow the GTF standard

Parameters

unsigned frame_height the total height of the frame (including blanking) in lines.

unsigned hfreq the horizontal frequency in Hz.

unsigned vsync the height of the vertical sync in lines.

u32 polarities the horizontal and vertical polarities (same as struct v4l2_bt_timings polarities).

bool interlaced if this flag is true, it indicates interlaced format

struct v4l2_fract aspect preferred aspect ratio. GTF has no method of determining the aspect ratio in order to derive the image width from the image height, so it has to be passed explicitly. Usually the native screen aspect ratio is used for this. If it is not filled in correctly, then 16:9 will be assumed.

struct v4l2_dv_timings * fmt the resulting timings.

Description

This function will attempt to detect if the given values correspond to a valid GTF format. If so, then it will return true, and fmt will be filled in with the found GTF timings.

struct *v4l2_fract* **v4l2_calc_aspect_ratio**(u8 *hor_landscape*, u8 *vert_portrait*)
calculate the aspect ratio based on bytes 0x15 and 0x16 from the EDID.

Parameters

u8 hor_landscape byte 0x15 from the EDID.

u8 vert_portrait byte 0x16 from the EDID.

Description

Determines the aspect ratio from the EDID. See VESA Enhanced EDID standard, release A, rev 2, section 3.6.2: “Horizontal and Vertical Screen Size or Aspect Ratio”

struct *v4l2_fract* **v4l2_dv_timings_aspect_ratio**(const struct *v4l2_dv_timings* * *t*)
calculate the aspect ratio based on the v4l2_dv_timings information.

Parameters

const struct v4l2_dv_timings * t the timings data.

2.1.17 V4L2 flash functions and data structures

struct **v4l2_flash_ctrl_data**
flash control initialization data, filled basing on the features declared by the LED flash class driver in the v4l2_flash_config

Definition

```
struct v4l2_flash_ctrl_data {
  struct v4l2_ctrl_config config;
  u32 cid;
};
```

Members

config initialization data for a control

cid contains v4l2 flash control id if the config field was initialized, 0 otherwise

struct **v4l2_flash_ops**
V4L2 flash operations

Definition

```
struct v4l2_flash_ops {
  int (* external_strobe_set) (struct v4l2_flash *v4l2_flash,bool enable);
  enum led_brightness (* intensity_to_led_brightness) (struct v4l2_flash *v4l2_flash, s32␣
 ↪intensity);
  s32 (* led_brightness_to_intensity) (struct v4l2_flash *v4l2_flash, enum led_brightness);
};
```

Members

external_strobe_set Setup strobing the flash by hardware pin state assertion.

intensity_to_led_brightness Convert intensity to brightness in a device specific manner

led_brightness_to_intensity convert brightness to intensity in a device specific manner.

struct **v4l2_flash_config**
V4L2 Flash sub-device initialization data

Definition

```
struct v4l2_flash_config {
  char dev_name[32];
  struct led_flash_setting torch_intensity;
  struct led_flash_setting indicator_intensity;
  u32 flash_faults;
  unsigned int has_external_strobe:1;
};
```

Members

dev_name[32] the name of the media entity, unique in the system

torch_intensity constraints for the LED in torch mode

indicator_intensity constraints for the indicator LED

flash_faults bitmask of flash faults that the LED flash class device can report; corresponding LED_FAULT* bit definitions are available in the header file <linux/led-class-flash.h>

has_external_strobe external strobe capability

struct **v4l2_flash**
Flash sub-device context

Definition

```
struct v4l2_flash {
  struct led_classdev_flash * fled_cdev;
  struct led_classdev_flash * iled_cdev;
  const struct v4l2_flash_ops * ops;
```

```
  struct v4l2_subdev sd;
  struct v4l2_ctrl_handler hdl;
  struct v4l2_ctrl ** ctrls;
};
```

Members

fled_cdev LED flash class device controlled by this sub-device

iled_cdev LED class device representing indicator LED associated with the LED flash class device

ops V4L2 specific flash ops

sd V4L2 sub-device

hdl flash controls handler

ctrls array of pointers to controls, whose values define the sub-device state

struct *v4l2_flash* * **v4l2_flash_init**(struct device * *dev*, struct device_node * *of_node*, struct led_classdev_flash * *fled_cdev*, struct led_classdev_flash * *iled_cdev*, const struct *v4l2_flash_ops* * *ops*, struct *v4l2_flash_config* * *config*)
initialize V4L2 flash led sub-device

Parameters

struct device * dev flash device, e.g. an I2C device

struct device_node * of_node of_node of the LED, may be NULL if the same as device's

struct led_classdev_flash * fled_cdev LED flash class device to wrap

struct led_classdev_flash * iled_cdev LED flash class device representing indicator LED associated with fled_cdev, may be NULL

const struct v4l2_flash_ops * ops V4L2 Flash device ops

struct v4l2_flash_config * config initialization data for V4L2 Flash sub-device

Description

Create V4L2 Flash sub-device wrapping given LED subsystem device.

Return

A valid pointer, or, when an error occurs, the return value is encoded using ERR_PTR(). Use IS_ERR() to check and PTR_ERR() to obtain the numeric return value.

void **v4l2_flash_release**(struct *v4l2_flash* * *v4l2_flash*)
release V4L2 Flash sub-device

Parameters

struct v4l2_flash * v4l2_flash the V4L2 Flash sub-device to release

Description

Release V4L2 Flash sub-device.

2.1.18 V4L2 Media Controller functions and data structures

enum **tuner_pad_index**
tuner pad index for MEDIA_ENT_F_TUNER

Constants

TUNER_PAD_RF_INPUT Radiofrequency (RF) sink pad, usually linked to a RF connector entity.

TUNER_PAD_OUTPUT Tuner video output source pad. Contains the video chrominance and luminance or the hole bandwidth of the signal converted to an Intermediate Frequency (IF) or to baseband (on zero-IF tuners).

TUNER_PAD_AUD_OUT Tuner audio output source pad. Tuners used to decode analog TV signals have an extra pad for audio output. Old tuners use an analog stage with a saw filter for the audio IF frequency. The output of the pad is, in this case, the audio IF, with should be decoded either by the bridge chipset (that's the case of cx2388x chipsets) or may require an external IF sound processor, like msp34xx. On modern silicon tuners, the audio IF decoder is usually incorporated at the tuner. On such case, the output of this pad is an audio sampled data.

TUNER_NUM_PADS Number of pads of the tuner.

enum **if_vid_dec_pad_index**
video IF-PLL pad index for MEDIA_ENT_F_IF_VID_DECODER

Constants

IF_VID_DEC_PAD_IF_INPUT video Intermediate Frequency (IF) sink pad

IF_VID_DEC_PAD_OUT IF-PLL video output source pad. Contains the video chrominance and luminance IF signals.

IF_VID_DEC_PAD_NUM_PADS Number of pads of the video IF-PLL.

enum **if_aud_dec_pad_index**
audio/sound IF-PLL pad index for MEDIA_ENT_F_IF_AUD_DECODER

Constants

IF_AUD_DEC_PAD_IF_INPUT audio Intermediate Frequency (IF) sink pad

IF_AUD_DEC_PAD_OUT IF-PLL audio output source pad. Contains the audio sampled stream data, usually connected to the bridge bus via an Inter-IC Sound (I2S) bus.

IF_AUD_DEC_PAD_NUM_PADS Number of pads of the audio IF-PLL.

enum **demod_pad_index**
analog TV pad index for MEDIA_ENT_F_ATV_DECODER

Constants

DEMOD_PAD_IF_INPUT IF input sink pad.

DEMOD_PAD_VID_OUT Video output source pad.

DEMOD_PAD_VBI_OUT Vertical Blank Interface (VBI) output source pad.

DEMOD_PAD_AUDIO_OUT Audio output source pad.

DEMOD_NUM_PADS Maximum number of output pads.

int **v4l2_mc_create_media_graph**(struct *media_device* * *mdev*)
create Media Controller links at the graph.

Parameters

struct media_device * mdev pointer to the *media_device* struct.

Description

Add links between the entities commonly found on PC customer's hardware at the V4L2 side: camera sensors, audio and video PLL-IF decoders, tuners, analog TV decoder and I/O entities (video, VBI and Software Defined Radio).

Note:

Webcams are modelled on a very simple way: the sensor is connected directly to the I/O entity. All dirty details, like scaler and crop HW are hidden. While such mapping is enough for v4l2 interface centric PC-consumer's hardware, V4L2 subdev centric camera hardware should not use this routine, as it will not build the right graph.

int **v4l_enable_media_source**(struct *video_device* * *vdev*)
Hold media source for exclusive use if free

Parameters

struct video_device * vdev pointer to struct video_device

Description

This interface calls enable_source handler to determine if media source is free for use. The enable_source handler is responsible for checking is the media source is free and start a pipeline between the media source and the media entity associated with the video device. This interface should be called from v4l2-core and dvb-core interfaces that change the source configuration.

Return

returns zero on success or a negative error code.

void **v4l_disable_media_source**(struct *video_device* * *vdev*)
Release media source

Parameters

struct video_device * vdev pointer to struct video_device

Description

This interface calls disable_source handler to release the media source. The disable_source handler stops the active media pipeline between the media source and the media entity associated with the video device.

Return

returns zero on success or a negative error code.

int **v4l2_pipeline_pm_use**(struct *media_entity* * *entity*, int *use*)
Update the use count of an entity

Parameters

struct media_entity * entity The entity

int use Use (1) or stop using (0) the entity

Description

Update the use count of all entities in the pipeline and power entities on or off accordingly.

This function is intended to be called in video node open (use == 1) and release (use == 0). It uses struct media_entity.use_count to track the power status. The use of this function should be paired with *v4l2_pipeline_link_notify()*.

Return 0 on success or a negative error code on failure. Powering entities off is assumed to never fail. No failure can occur when the use parameter is set to 0.

int **v4l2_pipeline_link_notify**(struct *media_link* * *link*, u32 *flags*, unsigned int *notification*)
Link management notification callback

Parameters

struct media_link * link The link

u32 flags New link flags that will be applied

unsigned int notification The link's state change notification type (MEDIA_DEV_NOTIFY_*)

Description

React to link management on powered pipelines by updating the use count of all entities in the source and sink sides of the link. Entities are powered on or off accordingly. The use of this function should be paired with *v4l2_pipeline_pm_use()*.

Return 0 on success or a negative error code on failure. Powering entities off is assumed to never fail. This function will not fail for disconnection events.

2.1.19 V4L2 Media Bus functions and data structures

enum **v4l2_mbus_type**
media bus type

Constants

V4L2_MBUS_PARALLEL parallel interface with hsync and vsync

V4L2_MBUS_BT656 parallel interface with embedded synchronisation, can also be used for BT.1120

V4L2_MBUS_CSI2 MIPI CSI-2 serial interface

struct **v4l2_mbus_config**
media bus configuration

Definition

```
struct v4l2_mbus_config {
  enum v4l2_mbus_type type;
  unsigned int flags;
};
```

Members

type in: interface type

flags in / out: configuration flags, depending on **type**

2.1.20 V4L2 Memory to Memory functions and data structures

struct **v4l2_m2m_ops**
mem-to-mem device driver callbacks

Definition

```
struct v4l2_m2m_ops {
  void (* device_run) (void *priv);
  int (* job_ready) (void *priv);
  void (* job_abort) (void *priv);
  void (* lock) (void *priv);
  void (* unlock) (void *priv);
};
```

Members

device_run required. Begin the actual job (transaction) inside this callback. The job does NOT have to end before this callback returns (and it will be the usual case). When the job finishes, *v4l2_m2m_job_finish()* has to be called.

job_ready optional. Should return 0 if the driver does not have a job fully prepared to run yet (i.e. it will not be able to finish a transaction without sleeping). If not provided, it will be assumed that one source and one destination buffer are all that is required for the driver to perform one full transaction. This method may not sleep.

job_abort required. Informs the driver that it has to abort the currently running transaction as soon as possible (i.e. as soon as it can stop the device safely; e.g. in the next interrupt handler), even if the transaction would not have been finished by then. After the driver performs the necessary steps, it has to call *v4l2_m2m_job_finish()* (as if the transaction ended normally). This function does not have to (and will usually not) wait until the device enters a state when it can be stopped.

lock optional. Define a driver's own lock callback, instead of using *v4l2_m2m_ctx->q_lock*.

unlock optional. Define a driver's own unlock callback, instead of using *v4l2_m2m_ctx->q_lock*.

struct **v4l2_m2m_queue_ctx**
represents a queue for buffers ready to be processed

Definition

```
struct v4l2_m2m_queue_ctx {
  struct vb2_queue q;
  struct list_head rdy_queue;
  spinlock_t rdy_spinlock;
  u8 num_rdy;
  bool buffered;
};
```

Members

q pointer to struct *vb2_queue*

rdy_queue List of V4L2 mem-to-mem queues

rdy_spinlock spin lock to protect the struct usage

num_rdy number of buffers ready to be processed

buffered is the queue buffered?

Description

Queue for buffers ready to be processed as soon as this instance receives access to the device.

struct **v4l2_m2m_ctx**
Memory to memory context structure

Definition

```
struct v4l2_m2m_ctx {
  struct mutex * q_lock;
  struct v4l2_m2m_dev * m2m_dev;
  struct v4l2_m2m_queue_ctx cap_q_ctx;
  struct v4l2_m2m_queue_ctx out_q_ctx;
  struct list_head queue;
  unsigned long job_flags;
  wait_queue_head_t finished;
  void * priv;
};
```

Members

q_lock struct mutex lock

m2m_dev opaque pointer to the internal data to handle M2M context

cap_q_ctx Capture (output to memory) queue context

out_q_ctx Output (input from memory) queue context

queue List of memory to memory contexts

job_flags Job queue flags, used internally by v4l2-mem2mem.c: TRANS_QUEUED, TRANS_RUNNING and TRANS_ABORT.

finished Wait queue used to signalize when a job queue finished.

priv Instance private data

Description

The memory to memory context is specific to a file handle, NOT to e.g. a device.

struct **v4l2_m2m_buffer**
Memory to memory buffer

Definition

```
struct v4l2_m2m_buffer {
  struct vb2_v4l2_buffer vb;
  struct list_head list;
};
```

Members

vb pointer to struct *vb2_v4l2_buffer*

list list of m2m buffers

void * **v4l2_m2m_get_curr_priv**(struct v4l2_m2m_dev * *m2m_dev*)
return driver private data for the currently running instance or NULL if no instance is running

Parameters

struct v4l2_m2m_dev * m2m_dev opaque pointer to the internal data to handle M2M context

struct *vb2_queue* * **v4l2_m2m_get_vq**(struct *v4l2_m2m_ctx* * *m2m_ctx*, enum *v4l2_buf_type type*)
return vb2_queue for the given type

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

enum v4l2_buf_type type type of the V4L2 buffer, as defined by enum *v4l2_buf_type*

void **v4l2_m2m_try_schedule**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
check whether an instance is ready to be added to the pending job queue and add it if so.

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

Description

There are three basic requirements an instance has to meet to be able to run: 1) at least one source buffer has to be queued, 2) at least one destination buffer has to be queued, 3) streaming has to be on.

If a queue is buffered (for example a decoder hardware ringbuffer that has to be drained before doing streamoff), allow scheduling without v4l2 buffers on that queue.

There may also be additional, custom requirements. In such case the driver should supply a custom callback (job_ready in v4l2_m2m_ops) that should return 1 if the instance is ready. An example of the above could be an instance that requires more than one src/dst buffer per transaction.

void **v4l2_m2m_job_finish**(struct v4l2_m2m_dev * *m2m_dev*, struct *v4l2_m2m_ctx* * *m2m_ctx*)
inform the framework that a job has been finished and have it clean up

Parameters

struct v4l2_m2m_dev * m2m_dev opaque pointer to the internal data to handle M2M context

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

Description

Called by a driver to yield back the device after it has finished with it. Should be called as soon as possible after reaching a state which allows other instances to take control of the device.

This function has to be called only after *v4l2_m2m_ops->device_run* callback has been called on the driver. To prevent recursion, it should not be called directly from the *v4l2_m2m_ops->device_run* callback though.

int **v4l2_m2m_reqbufs**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, struct *v4l2_requestbuffers* * *reqbufs*)
multi-queue-aware REQBUFS multiplexer

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct v4l2_requestbuffers * reqbufs pointer to struct *v4l2_requestbuffers*

int **v4l2_m2m_querybuf**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, struct *v4l2_buffer* * *buf*)
multi-queue-aware QUERYBUF multiplexer

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct v4l2_buffer * buf pointer to struct *v4l2_buffer*

Description

See *v4l2_m2m_mmap()* documentation for details.

int **v4l2_m2m_qbuf**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, struct *v4l2_buffer* * *buf*)
enqueue a source or destination buffer, depending on the type

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct v4l2_buffer * buf pointer to struct *v4l2_buffer*

int **v4l2_m2m_dqbuf**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, struct *v4l2_buffer* * *buf*)
dequeue a source or destination buffer, depending on the type

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct v4l2_buffer * buf pointer to struct *v4l2_buffer*

int **v4l2_m2m_prepare_buf**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, struct *v4l2_buffer* * *buf*)
prepare a source or destination buffer, depending on the type

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct v4l2_buffer * buf pointer to struct *v4l2_buffer*

int **v4l2_m2m_create_bufs**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, struct *v4l2_create_buffers* * *create*)
create a source or destination buffer, depending on the type

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct v4l2_create_buffers * create pointer to struct *v4l2_create_buffers*

int **v4l2_m2m_expbuf**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, struct *v4l2_exportbuffer* * *eb*)
export a source or destination buffer, depending on the type

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct v4l2_exportbuffer * eb pointer to struct *v4l2_exportbuffer*

int **v4l2_m2m_streamon**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, enum *v4l2_buf_type type*)
turn on streaming for a video queue

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

enum v4l2_buf_type type type of the V4L2 buffer, as defined by enum *v4l2_buf_type*

int **v4l2_m2m_streamoff**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, enum *v4l2_buf_type type*)
turn off streaming for a video queue

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

enum v4l2_buf_type type type of the V4L2 buffer, as defined by enum *v4l2_buf_type*

unsigned int **v4l2_m2m_poll**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, struct poll_table_struct * *wait*)
poll replacement, for destination buffers only

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct poll_table_struct * wait pointer to struct poll_table_struct

Description

Call from the driver's poll() function. Will poll both queues. If a buffer is available to dequeue (with dqbuf) from the source queue, this will indicate that a non-blocking write can be performed, while read will be returned in case of the destination queue.

int **v4l2_m2m_mmap**(struct file * *file*, struct *v4l2_m2m_ctx* * *m2m_ctx*, struct vm_area_struct * *vma*)
source and destination queues-aware mmap multiplexer

Parameters

struct file * file pointer to struct file

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct vm_area_struct * vma pointer to struct vm_area_struct

Description

Call from driver's `mmap()` function. Will handle `mmap()` for both queues seamlessly for videobuffer, which will receive normal per-queue offsets and proper videobuf queue pointers. The differentiation is made outside videobuf by adding a predefined offset to buffers from one of the queues and subtracting it before passing it back to videobuf. Only drivers (and thus applications) receive modified offsets.

struct v4l2_m2m_dev * **v4l2_m2m_init**(const struct *v4l2_m2m_ops* * *m2m_ops*)
initialize per-driver m2m data

Parameters

const struct v4l2_m2m_ops * m2m_ops pointer to struct v4l2_m2m_ops

Description

Usually called from driver's :c:func:`probe()` function.

Return

returns an opaque pointer to the internal data to handle M2M context

void **v4l2_m2m_release**(struct v4l2_m2m_dev * *m2m_dev*)
cleans up and frees a m2m_dev structure

Parameters

struct v4l2_m2m_dev * m2m_dev opaque pointer to the internal data to handle M2M context

Description

Usually called from driver's :c:func:`remove()` function.

struct *v4l2_m2m_ctx* * **v4l2_m2m_ctx_init**(struct v4l2_m2m_dev * *m2m_dev*, void * *drv_priv*, int (*queue_init) (void *priv, struct *vb2_queue* *src_vq, struct *vb2_queue* *dst_vq))
allocate and initialize a m2m context

Parameters

struct v4l2_m2m_dev * m2m_dev opaque pointer to the internal data to handle M2M context

void * drv_priv driver's instance private data

int (*)(void *priv,struct vb2_queue *src_vq,struct vb2_queue *dst_vq) queue_init a callback for queue type-specific initialization function to be used for initializing videobuf_queues

Description

Usually called from driver's :c:func:`open()` function.

void **v4l2_m2m_ctx_release**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
release m2m context

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

Description

Usually called from driver's `release()` function.

void **v4l2_m2m_buf_queue**(struct *v4l2_m2m_ctx* * *m2m_ctx*, struct *vb2_v4l2_buffer* * *vbuf*)
add a buffer to the proper ready buffers list.

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct vb2_v4l2_buffer * vbuf pointer to struct *vb2_v4l2_buffer*

Description

Call from videobuf_queue_ops->ops->buf_queue, videobuf_queue_ops callback.

unsigned int **v4l2_m2m_num_src_bufs_ready**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
return the number of source buffers ready for use

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

unsigned int **v4l2_m2m_num_dst_bufs_ready**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
return the number of destination buffers ready for use

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

void * **v4l2_m2m_next_buf**(struct *v4l2_m2m_queue_ctx* * *q_ctx*)
return next buffer from the list of ready buffers

Parameters

struct v4l2_m2m_queue_ctx * q_ctx pointer to struct **v4l2_m2m_queue_ctx**

void * **v4l2_m2m_next_src_buf**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
return next source buffer from the list of ready buffers

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

void * **v4l2_m2m_next_dst_buf**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
return next destination buffer from the list of ready buffers

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_queue* * **v4l2_m2m_get_src_vq**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
return vb2_queue for source buffers

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_queue* * **v4l2_m2m_get_dst_vq**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
return vb2_queue for destination buffers

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

void * **v4l2_m2m_buf_remove**(struct *v4l2_m2m_queue_ctx* * *q_ctx*)
take off a buffer from the list of ready buffers and return it

Parameters

struct v4l2_m2m_queue_ctx * q_ctx pointer to struct **v4l2_m2m_queue_ctx**

void * **v4l2_m2m_src_buf_remove**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
take off a source buffer from the list of ready buffers and return it

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

void * **v4l2_m2m_dst_buf_remove**(struct *v4l2_m2m_ctx* * *m2m_ctx*)
take off a destination buffer from the list of ready buffers and return it

Parameters

struct v4l2_m2m_ctx * m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

2.1.21 V4L2 Open Firmware kAPI

struct **v4l2_of_bus_mipi_csi2**
MIPI CSI-2 bus data structure

Definition

```
struct v4l2_of_bus_mipi_csi2 {
  unsigned int flags;
  unsigned char data_lanes[4];
  unsigned char clock_lane;
  unsigned short num_data_lanes;
  bool lane_polarities[5];
};
```

Members

flags media bus (V4L2_MBUS_*) flags

data_lanes[4] an array of physical data lane indexes

clock_lane physical lane index of the clock lane

num_data_lanes number of data lanes

lane_polarities[5] polarity of the lanes. The order is the same of the physical lanes.

struct **v4l2_of_bus_parallel**
parallel data bus data structure

Definition

```
struct v4l2_of_bus_parallel {
  unsigned int flags;
  unsigned char bus_width;
  unsigned char data_shift;
};
```

Members

flags media bus (V4L2_MBUS_*) flags

bus_width bus width in bits

data_shift data shift in bits

struct **v4l2_of_endpoint**
the endpoint data structure

Definition

```
struct v4l2_of_endpoint {
  struct of_endpoint base;
  enum v4l2_mbus_type bus_type;
  union bus;
  u64 * link_frequencies;
  unsigned int nr_of_link_frequencies;
};
```

Members

base struct of_endpoint containing port, id, and local of_node

bus_type bus type

bus bus configuration data structure

link_frequencies array of supported link frequencies

nr_of_link_frequencies number of elements in link_frequenccies array

struct **v4l2_of_link**
a link between two endpoints

Definition

```
struct v4l2_of_link {
  struct device_node * local_node;
  unsigned int local_port;
  struct device_node * remote_node;
  unsigned int remote_port;
};
```

Members

local_node pointer to device_node of this endpoint

local_port identifier of the port this endpoint belongs to

remote_node pointer to device_node of the remote endpoint

remote_port identifier of the port the remote endpoint belongs to

2.1.22 V4L2 rect helper functions

void **v4l2_rect_set_size_to**(struct *v4l2_rect * r*, const struct *v4l2_rect * size*)
copy the width/height values.

Parameters

struct v4l2_rect * r rect whose width and height fields will be set

const struct v4l2_rect * size rect containing the width and height fields you need.

void **v4l2_rect_set_min_size**(struct *v4l2_rect * r*, const struct *v4l2_rect * min_size*)
width and height of r should be >= min_size.

Parameters

struct v4l2_rect * r rect whose width and height will be modified

const struct v4l2_rect * min_size rect containing the minimal width and height

void **v4l2_rect_set_max_size**(struct *v4l2_rect * r*, const struct *v4l2_rect * max_size*)
width and height of r should be <= max_size

Parameters

struct v4l2_rect * r rect whose width and height will be modified

const struct v4l2_rect * max_size rect containing the maximum width and height

void **v4l2_rect_map_inside**(struct *v4l2_rect * r*, const struct *v4l2_rect * boundary*)
r should be inside boundary.

Parameters

struct v4l2_rect * r rect that will be modified

const struct v4l2_rect * boundary rect containing the boundary for **r**

bool **v4l2_rect_same_size**(const struct *v4l2_rect * r1*, const struct *v4l2_rect * r2*)
return true if r1 has the same size as r2

Parameters

const struct v4l2_rect * r1 rectangle.

const struct v4l2_rect * r2 rectangle.

Description

Return true if both rectangles have the same size.

void **v4l2_rect_intersect**(struct *v4l2_rect * r*, const struct *v4l2_rect * r1*, const struct *v4l2_rect * r2*)
calculate the intersection of two rects.

Parameters

struct v4l2_rect * r intersection of **r1** and **r2**.

const struct v4l2_rect * r1 rectangle.

const struct v4l2_rect * r2 rectangle.

void **v4l2_rect_scale**(struct *v4l2_rect * r*, const struct *v4l2_rect * from*, const struct *v4l2_rect * to*)
scale rect r by to/from

Parameters

struct v4l2_rect * r rect to be scaled.

const struct v4l2_rect * from from rectangle.

const struct v4l2_rect * to to rectangle.

Description

This scales rectangle **r** horizontally by **to**->width / **from**->width and vertically by **to**->height / **from**->height.

Typically **r** is a rectangle inside **from** and you want the rectangle as it would appear after scaling **from** to **to**. So the resulting **r** will be the scaled rectangle inside **to**.

bool **v4l2_rect_overlap**(const struct *v4l2_rect * r1*, const struct *v4l2_rect * r2*)
do r1 and r2 overlap?

Parameters

const struct v4l2_rect * r1 rectangle.

const struct v4l2_rect * r2 rectangle.

Description

Returns true if **r1** and **r2** overlap.

2.1.23 Tuner functions and data structures

enum **tuner_mode**
Mode of the tuner

Constants

T_RADIO Tuner core will work in radio mode

T_ANALOG_TV Tuner core will work in analog TV mode

Description

Older boards only had a single tuner device, but some devices have a separate tuner for radio. In any case, the tuner-core needs to know if the tuner chip(s) will be used in radio mode or analog TV mode, as, on radio mode, frequencies are specified on a different range than on TV mode. This enum is used by the tuner core in order to work with the proper tuner range and eventually use a different tuner chip while in radio mode.

struct **tuner_setup**
setup the tuner chipsets

Definition

```
struct tuner_setup {
  unsigned short addr;
  unsigned int type;
  unsigned int mode_mask;
  void * config;
  int (* tuner_callback) (void *dev, int component, int cmd, int arg);
};
```

Members

addr I2C address used to control the tuner device/chipset

type Type of the tuner, as defined at the TUNER_* macros. Each different tuner model should have an unique identifier.

mode_mask Mask with the allowed tuner modes: V4L2_TUNER_RADIO, V4L2_TUNER_ANALOG_TV and/or V4L2_TUNER_DIGITAL_TV, describing if the tuner should be used to support Radio, analog TV and/or digital TV.

config Used to send tuner-specific configuration for complex tuners that require extra parameters to be set. Only a very few tuners require it and its usage on newer tuners should be avoided.

tuner_callback Some tuners require to call back the bridge driver, in order to do some tasks like rising a GPIO at the bridge chipset, in order to do things like resetting the device.

Description

Older boards only had a single tuner device. Nowadays multiple tuner devices may be present on a single board. Using TUNER_SET_TYPE_ADDR to pass the tuner_setup structure it is possible to setup each tuner device in turn.

Since multiple devices may be present it is no longer sufficient to send a command to a single i2c device. Instead you should broadcast the command to all i2c devices.

By setting the mode_mask correctly you can select which commands are accepted by a specific tuner device. For example, set mode_mask to T_RADIO if the device is a radio-only tuner. That specific tuner will only accept commands when the tuner is in radio mode and ignore them when the tuner is set to TV mode.

enum **param_type**
type of the tuner pameters

Constants

TUNER_PARAM_TYPE_RADIO Tuner params are for FM and/or AM radio

TUNER_PARAM_TYPE_PAL Tuner params are for PAL color TV standard

TUNER_PARAM_TYPE_SECAM Tuner params are for SECAM color TV standard

TUNER_PARAM_TYPE_NTSC Tuner params are for NTSC color TV standard

TUNER_PARAM_TYPE_DIGITAL Tuner params are for digital TV

struct **tuner_range**
define the frequencies supported by the tuner

Definition

```
struct tuner_range {
  unsigned short limit;
  unsigned char config;
  unsigned char cb;
};
```

Members

limit Max frequency supported by that range, in 62.5 kHz (TV) or 62.5 Hz (Radio), as defined by V4L2_TUNER_CAP_LOW.

config Value of the band switch byte (BB) to setup this mode.

cb Value of the CB byte to setup this mode.

Description

Please notice that digital tuners like xc3028/xc4000/xc5000 don't use those ranges, as they're defined inside the driver. This is used by analog tuners that are compatible with the "Philips way" to setup the tuners. On those devices, the tuner set is done via 4 bytes:

1. divider byte1 (DB1)
2. divider byte 2 (DB2)
3. Control byte (CB)
4. band switch byte (BB)

Some tuners also have an additional optional Auxiliary byte (AB).

struct **tuner_params**
Parameters to be used to setup the tuner. Those are used by drivers/media/tuners/tuner-types.c in order to specify the tuner properties. Most of the parameters are for tuners based on tda9887 IF-PLL multi-standard analog TV/Radio demodulator, with is very common on legacy analog tuners.

Definition

```
struct tuner_params {
  enum param_type type;
  unsigned int cb_first_if_lower_freq:1;
  unsigned int has_tda9887:1;
  unsigned int port1_fm_high_sensitivity:1;
  unsigned int port2_fm_high_sensitivity:1;
  unsigned int fm_gain_normal:1;
  unsigned int intercarrier_mode:1;
  unsigned int port1_active:1;
  unsigned int port2_active:1;
  unsigned int port1_invert_for_secam_lc:1;
  unsigned int port2_invert_for_secam_lc:1;
  unsigned int port1_set_for_fm_mono:1;
  unsigned int default_pll_gating_18:1;
  unsigned int radio_if:2;
  signed int default_top_low:5;
  signed int default_top_mid:5;
  signed int default_top_high:5;
  signed int default_top_secam_low:5;
  signed int default_top_secam_mid:5;
  signed int default_top_secam_high:5;
  u16 iffreq;
  unsigned int count;
  struct tuner_range * ranges;
};
```

Members

type Type of the tuner parameters, as defined at enum param_type. If the tuner supports multiple standards, an array should be used, with one row per different standard.

cb_first_if_lower_freq Many Philips-based tuners have a comment in their datasheet like "For channel selection involving band switching, and to ensure smooth tuning to the desired channel without causing unnecessary charge pump action, it is recommended to consider the difference between wanted channel frequency and the current channel frequency. Unnecessary charge pump action will result in very low tuning voltage which may drive the oscillator to extreme conditions". Set cb_first_if_lower_freq to 1, if this check is required for this tuner. I tested this for PAL by first setting

the TV frequency to 203 MHz and then switching to 96.6 MHz FM radio. The result was static unless the control byte was sent first.

has_tda9887 Set to 1 if this tuner uses a tda9887

port1_fm_high_sensitivity Many Philips tuners use tda9887 PORT1 to select the FM radio sensitivity. If this setting is 1, then set PORT1 to 1 to get proper FM reception.

port2_fm_high_sensitivity Some Philips tuners use tda9887 PORT2 to select the FM radio sensitivity. If this setting is 1, then set PORT2 to 1 to get proper FM reception.

fm_gain_normal Some Philips tuners use tda9887 cGainNormal to select the FM radio sensitivity. If this setting is 1, e register will use cGainNormal instead of cGainLow.

intercarrier_mode Most tuners with a tda9887 use QSS mode. Some (cheaper) tuners use Intercarrier mode. If this setting is 1, then the tuner needs to be set to intercarrier mode.

port1_active This setting sets the default value for PORT1. 0 means inactive, 1 means active. Note: the actual bit value written to the tda9887 is inverted. So a 0 here means a 1 in the B6 bit.

port2_active This setting sets the default value for PORT2. 0 means inactive, 1 means active. Note: the actual bit value written to the tda9887 is inverted. So a 0 here means a 1 in the B7 bit.

port1_invert_for_secam_lc Sometimes PORT1 is inverted when the SECAM-L' standard is selected. Set this bit to 1 if this is needed.

port2_invert_for_secam_lc Sometimes PORT2 is inverted when the SECAM-L' standard is selected. Set this bit to 1 if this is needed.

port1_set_for_fm_mono Some cards require PORT1 to be 1 for mono Radio FM and 0 for stereo.

default_pll_gating_18 Select 18% (or according to datasheet 0%) L standard PLL gating, vs the driver default of 36%.

radio_if IF to use in radio mode. Tuners with a separate radio IF filter seem to use 10.7, while those without use 33.3 for PAL/SECAM tuners and 41.3 for NTSC tuners. 0 = 10.7, 1 = 33.3, 2 = 41.3

default_top_low Default tda9887 TOP value in dB for the low band. Default is 0. Range: -16:+15

default_top_mid Default tda9887 TOP value in dB for the mid band. Default is 0. Range: -16:+15

default_top_high Default tda9887 TOP value in dB for the high band. Default is 0. Range: -16:+15

default_top_secam_low Default tda9887 TOP value in dB for SECAM-L/L' for the low band. Default is 0. Several tuners require a different TOP value for the SECAM-L/L' standards. Range: -16:+15

default_top_secam_mid Default tda9887 TOP value in dB for SECAM-L/L' for the mid band. Default is 0. Several tuners require a different TOP value for the SECAM-L/L' standards. Range: -16:+15

default_top_secam_high Default tda9887 TOP value in dB for SECAM-L/L' for the high band. Default is 0. Several tuners require a different TOP value for the SECAM-L/L' standards. Range: -16:+15

iffreq Intermediate frequency (IF) used by the tuner on digital mode.

count Size of the ranges array.

ranges Array with the frequency ranges supported by the tuner.

2.1.24 V4L2 common functions and data structures

int **v4l2_ctrl_query_fill**(struct *v4l2_queryctrl* * *qctrl*, s32 *min*, s32 *max*, s32 *step*, s32 *def*)
Fill in a struct v4l2_queryctrl

Parameters

struct v4l2_queryctrl * qctrl pointer to the *struct v4l2_queryctrl* to be filled

s32 min minimum value for the control

s32 max maximum value for the control

s32 step control step

s32 def default value for the control

Description

Fills the *struct v4l2_queryctrl* fields for the query control.

Note:

This function assumes that the ***qctrl->****id field is filled.*

Returns -EINVAL if the control is not known by the V4L2 core, 0 on success.

struct *v4l2_subdev* * **v4l2_i2c_new_subdev**(struct *v4l2_device* * *v4l2_dev*, struct i2c_adapter * *adapter*, const char * *client_type*, u8 *addr*, const unsigned short * *probe_addrs*)

Load an i2c module and return an initialized *struct v4l2_subdev*.

Parameters

struct v4l2_device * v4l2_dev pointer to *struct v4l2_device*

struct i2c_adapter * adapter pointer to struct i2c_adapter

const char * client_type name of the chip that's on the adapter.

u8 addr I2C address. If zero, it will use **probe_addrs**

const unsigned short * probe_addrs array with a list of address. The last entry at such array should be I2C_CLIENT_END.

Description

returns a *struct v4l2_subdev* pointer.

struct *v4l2_subdev* * **v4l2_i2c_new_subdev_board**(struct *v4l2_device* * *v4l2_dev*, struct i2c_adapter * *adapter*, struct i2c_board_info * *info*, const unsigned short * *probe_addrs*)

Load an i2c module and return an initialized *struct v4l2_subdev*.

Parameters

struct v4l2_device * v4l2_dev pointer to *struct v4l2_device*

struct i2c_adapter * adapter pointer to struct i2c_adapter

struct i2c_board_info * info pointer to struct i2c_board_info used to replace the irq, platform_data and addr arguments.

const unsigned short * probe_addrs array with a list of address. The last entry at such array should be I2C_CLIENT_END.

Description

returns a *struct v4l2_subdev* pointer.

void **v4l2_i2c_subdev_init**(struct *v4l2_subdev* * *sd*, struct i2c_client * *client*, const struct *v4l2_subdev_ops* * *ops*)

Initializes a *struct v4l2_subdev* with data from an i2c_client struct.

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

struct i2c_client * client pointer to struct i2c_client

const struct v4l2_subdev_ops * ops pointer to *struct v4l2_subdev_ops*

unsigned short **v4l2_i2c_subdev_addr**(struct *v4l2_subdev* * *sd*)
returns i2c client address of *struct v4l2_subdev*.

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

Description

Returns the address of an I2C sub-device

struct *v4l2_subdev* * **v4l2_spi_new_subdev**(struct *v4l2_device* * *v4l2_dev*, struct spi_master * *master*, struct spi_board_info * *info*)
Load an spi module and return an initialized *struct v4l2_subdev*.

Parameters

struct v4l2_device * v4l2_dev pointer to *struct v4l2_device*.

struct spi_master * master pointer to struct spi_master.

struct spi_board_info * info pointer to struct spi_board_info.

Description

returns a *struct v4l2_subdev* pointer.

void **v4l2_spi_subdev_init**(struct *v4l2_subdev* * *sd*, struct spi_device * *spi*, const struct *v4l2_subdev_ops* * *ops*)
Initialize a v4l2_subdev with data from an spi_device struct.

Parameters

struct v4l2_subdev * sd pointer to *struct v4l2_subdev*

struct spi_device * spi pointer to struct spi_device.

const struct v4l2_subdev_ops * ops pointer to *struct v4l2_subdev_ops*

struct **v4l2_ioctl_ops**
describe operations for each V4L2 ioctl

Definition

```
struct v4l2_ioctl_ops {
  int (* vidioc_querycap) (struct file *file, void *fh,struct v4l2_capability *cap);
  int (* vidioc_enum_fmt_vid_cap) (struct file *file, void *fh,struct v4l2_fmtdesc *f);
  int (* vidioc_enum_fmt_vid_overlay) (struct file *file, void *fh,struct v4l2_fmtdesc *f);
  int (* vidioc_enum_fmt_vid_out) (struct file *file, void *fh,struct v4l2_fmtdesc *f);
  int (* vidioc_enum_fmt_vid_cap_mplane) (struct file *file, void *fh,struct v4l2_fmtdesc *f);
  int (* vidioc_enum_fmt_vid_out_mplane) (struct file *file, void *fh,struct v4l2_fmtdesc *f);
  int (* vidioc_enum_fmt_sdr_cap) (struct file *file, void *fh,struct v4l2_fmtdesc *f);
  int (* vidioc_enum_fmt_sdr_out) (struct file *file, void *fh,struct v4l2_fmtdesc *f);
  int (* vidioc_g_fmt_vid_cap) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_vid_overlay) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_vid_out) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_vid_out_overlay) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_vbi_cap) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_vbi_out) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_sliced_vbi_cap) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_sliced_vbi_out) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_vid_cap_mplane) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_vid_out_mplane) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_sdr_cap) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_g_fmt_sdr_out) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_s_fmt_vid_cap) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_s_fmt_vid_overlay) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_s_fmt_vid_out) (struct file *file, void *fh,struct v4l2_format *f);
  int (* vidioc_s_fmt_vid_out_overlay) (struct file *file, void *fh,struct v4l2_format *f);
```

```
int (* vidioc_s_fmt_vbi_cap) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_s_fmt_vbi_out) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_s_fmt_sliced_vbi_cap) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_s_fmt_sliced_vbi_out) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_s_fmt_vid_cap_mplane) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_s_fmt_vid_out_mplane) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_s_fmt_sdr_cap) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_s_fmt_sdr_out) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_vid_cap) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_vid_overlay) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_vid_out) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_vid_out_overlay) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_vbi_cap) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_vbi_out) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_sliced_vbi_cap) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_sliced_vbi_out) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_vid_cap_mplane) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_vid_out_mplane) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_sdr_cap) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_try_fmt_sdr_out) (struct file *file, void *fh,struct v4l2_format *f);
int (* vidioc_reqbufs) (struct file *file, void *fh,struct v4l2_requestbuffers *b);
int (* vidioc_querybuf) (struct file *file, void *fh,struct v4l2_buffer *b);
int (* vidioc_qbuf) (struct file *file, void *fh,struct v4l2_buffer *b);
int (* vidioc_expbuf) (struct file *file, void *fh,struct v4l2_exportbuffer *e);
int (* vidioc_dqbuf) (struct file *file, void *fh,struct v4l2_buffer *b);
int (* vidioc_create_bufs) (struct file *file, void *fh,struct v4l2_create_buffers *b);
int (* vidioc_prepare_buf) (struct file *file, void *fh,struct v4l2_buffer *b);
int (* vidioc_overlay) (struct file *file, void *fh, unsigned int i);
int (* vidioc_g_fbuf) (struct file *file, void *fh,struct v4l2_framebuffer *a);
int (* vidioc_s_fbuf) (struct file *file, void *fh,const struct v4l2_framebuffer *a);
int (* vidioc_streamon) (struct file *file, void *fh,enum v4l2_buf_type i);
int (* vidioc_streamoff) (struct file *file, void *fh,enum v4l2_buf_type i);
int (* vidioc_g_std) (struct file *file, void *fh, v4l2_std_id *norm);
int (* vidioc_s_std) (struct file *file, void *fh, v4l2_std_id norm);
int (* vidioc_querystd) (struct file *file, void *fh, v4l2_std_id *a);
int (* vidioc_enum_input) (struct file *file, void *fh,struct v4l2_input *inp);
int (* vidioc_g_input) (struct file *file, void *fh, unsigned int *i);
int (* vidioc_s_input) (struct file *file, void *fh, unsigned int i);
int (* vidioc_enum_output) (struct file *file, void *fh,struct v4l2_output *a);
int (* vidioc_g_output) (struct file *file, void *fh, unsigned int *i);
int (* vidioc_s_output) (struct file *file, void *fh, unsigned int i);
int (* vidioc_queryctrl) (struct file *file, void *fh,struct v4l2_queryctrl *a);
int (* vidioc_query_ext_ctrl) (struct file *file, void *fh,struct v4l2_query_ext_ctrl *a);
int (* vidioc_g_ctrl) (struct file *file, void *fh,struct v4l2_control *a);
int (* vidioc_s_ctrl) (struct file *file, void *fh,struct v4l2_control *a);
int (* vidioc_g_ext_ctrls) (struct file *file, void *fh,struct v4l2_ext_controls *a);
int (* vidioc_s_ext_ctrls) (struct file *file, void *fh,struct v4l2_ext_controls *a);
int (* vidioc_try_ext_ctrls) (struct file *file, void *fh,struct v4l2_ext_controls *a);
int (* vidioc_querymenu) (struct file *file, void *fh,struct v4l2_querymenu *a);
int (* vidioc_enumaudio) (struct file *file, void *fh,struct v4l2_audio *a);
int (* vidioc_g_audio) (struct file *file, void *fh,struct v4l2_audio *a);
int (* vidioc_s_audio) (struct file *file, void *fh,const struct v4l2_audio *a);
int (* vidioc_enumaudout) (struct file *file, void *fh,struct v4l2_audioout *a);
int (* vidioc_g_audout) (struct file *file, void *fh,struct v4l2_audioout *a);
int (* vidioc_s_audout) (struct file *file, void *fh,const struct v4l2_audioout *a);
int (* vidioc_g_modulator) (struct file *file, void *fh,struct v4l2_modulator *a);
int (* vidioc_s_modulator) (struct file *file, void *fh,const struct v4l2_modulator *a);
int (* vidioc_cropcap) (struct file *file, void *fh,struct v4l2_cropcap *a);
int (* vidioc_g_crop) (struct file *file, void *fh,struct v4l2_crop *a);
int (* vidioc_s_crop) (struct file *file, void *fh,const struct v4l2_crop *a);
int (* vidioc_g_selection) (struct file *file, void *fh,struct v4l2_selection *s);
int (* vidioc_s_selection) (struct file *file, void *fh,struct v4l2_selection *s);
int (* vidioc_g_jpegcomp) (struct file *file, void *fh,struct v4l2_jpegcompression *a);
```

```
  int (* vidioc_s_jpegcomp) (struct file *file, void *fh,const struct v4l2_jpegcompression *a);
  int (* vidioc_g_enc_index) (struct file *file, void *fh,struct v4l2_enc_idx *a);
  int (* vidioc_encoder_cmd) (struct file *file, void *fh,struct v4l2_encoder_cmd *a);
  int (* vidioc_try_encoder_cmd) (struct file *file, void *fh,struct v4l2_encoder_cmd *a);
  int (* vidioc_decoder_cmd) (struct file *file, void *fh,struct v4l2_decoder_cmd *a);
  int (* vidioc_try_decoder_cmd) (struct file *file, void *fh,struct v4l2_decoder_cmd *a);
  int (* vidioc_g_parm) (struct file *file, void *fh,struct v4l2_streamparm *a);
  int (* vidioc_s_parm) (struct file *file, void *fh,struct v4l2_streamparm *a);
  int (* vidioc_g_tuner) (struct file *file, void *fh,struct v4l2_tuner *a);
  int (* vidioc_s_tuner) (struct file *file, void *fh,const struct v4l2_tuner *a);
  int (* vidioc_g_frequency) (struct file *file, void *fh,struct v4l2_frequency *a);
  int (* vidioc_s_frequency) (struct file *file, void *fh,const struct v4l2_frequency *a);
  int (* vidioc_enum_freq_bands) (struct file *file, void *fh,struct v4l2_frequency_band *band);
  int (* vidioc_g_sliced_vbi_cap) (struct file *file, void *fh,struct v4l2_sliced_vbi_cap *a);
  int (* vidioc_log_status) (struct file *file, void *fh);
  int (* vidioc_s_hw_freq_seek) (struct file *file, void *fh,const struct v4l2_hw_freq_seek *a);
#ifdef CONFIG_VIDEO_ADV_DEBUG
  int (* vidioc_g_register) (struct file *file, void *fh,struct v4l2_dbg_register *reg);
  int (* vidioc_s_register) (struct file *file, void *fh,const struct v4l2_dbg_register *reg);
  int (* vidioc_g_chip_info) (struct file *file, void *fh,struct v4l2_dbg_chip_info *chip);
#endif
  int (* vidioc_enum_framesizes) (struct file *file, void *fh,struct v4l2_frmsizeenum *fsize);
  int (* vidioc_enum_frameintervals) (struct file *file, void *fh,struct v4l2_frmivalenum␣
 ↪*fival);
  int (* vidioc_s_dv_timings) (struct file *file, void *fh,struct v4l2_dv_timings *timings);
  int (* vidioc_g_dv_timings) (struct file *file, void *fh,struct v4l2_dv_timings *timings);
  int (* vidioc_query_dv_timings) (struct file *file, void *fh,struct v4l2_dv_timings *timings);
  int (* vidioc_enum_dv_timings) (struct file *file, void *fh,struct v4l2_enum_dv_timings␣
 ↪*timings);
  int (* vidioc_dv_timings_cap) (struct file *file, void *fh,struct v4l2_dv_timings_cap *cap);
  int (* vidioc_g_edid) (struct file *file, void *fh,struct v4l2_edid *edid);
  int (* vidioc_s_edid) (struct file *file, void *fh,struct v4l2_edid *edid);
  int (* vidioc_subscribe_event) (struct v4l2_fh *fh,const struct v4l2_event_subscription *sub);
  int (* vidioc_unsubscribe_event) (struct v4l2_fh *fh,const struct v4l2_event_subscription␣
 ↪*sub);
  long (* vidioc_default) (struct file *file, void *fh,bool valid_prio, unsigned int cmd, void␣
 ↪*arg);
};
```

Members

vidioc_querycap pointer to the function that implements *VIDIOC_QUERYCAP* ioctl

vidioc_enum_fmt_vid_cap pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for video capture in single plane mode

vidioc_enum_fmt_vid_overlay pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for video overlay

vidioc_enum_fmt_vid_out pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for video output in single plane mode

vidioc_enum_fmt_vid_cap_mplane pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for video capture in multiplane mode

vidioc_enum_fmt_vid_out_mplane pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for video output in multiplane mode

vidioc_enum_fmt_sdr_cap pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for Software Defined Radio capture

vidioc_enum_fmt_sdr_out pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for Software Defined Radio output

vidioc_g_fmt_vid_cap pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video capture in single plane mode

vidioc_g_fmt_vid_overlay pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video overlay

vidioc_g_fmt_vid_out pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video out in single plane mode

vidioc_g_fmt_vid_out_overlay pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video overlay output

vidioc_g_fmt_vbi_cap pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for raw VBI capture

vidioc_g_fmt_vbi_out pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for raw VBI output

vidioc_g_fmt_sliced_vbi_cap pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for sliced VBI capture

vidioc_g_fmt_sliced_vbi_out pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for sliced VBI output

vidioc_g_fmt_vid_cap_mplane pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video capture in multiple plane mode

vidioc_g_fmt_vid_out_mplane pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video out in multiplane plane mode

vidioc_g_fmt_sdr_cap pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for Software Defined Radio capture

vidioc_g_fmt_sdr_out pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for Software Defined Radio output

vidioc_s_fmt_vid_cap pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video capture in single plane mode

vidioc_s_fmt_vid_overlay pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video overlay

vidioc_s_fmt_vid_out pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video out in single plane mode

vidioc_s_fmt_vid_out_overlay pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video overlay output

vidioc_s_fmt_vbi_cap pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for raw VBI capture

vidioc_s_fmt_vbi_out pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for raw VBI output

vidioc_s_fmt_sliced_vbi_cap pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for sliced VBI capture

vidioc_s_fmt_sliced_vbi_out pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for sliced VBI output

vidioc_s_fmt_vid_cap_mplane pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video capture in multiple plane mode

vidioc_s_fmt_vid_out_mplane pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video out in multiplane plane mode

vidioc_s_fmt_sdr_cap pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for Software Defined Radio capture

vidioc_s_fmt_sdr_out pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for Software Defined Radio output

vidioc_try_fmt_vid_cap pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video capture in single plane mode

vidioc_try_fmt_vid_overlay pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video overlay

vidioc_try_fmt_vid_out pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video out in single plane mode

vidioc_try_fmt_vid_out_overlay pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video overlay output

vidioc_try_fmt_vbi_cap pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for raw VBI capture

vidioc_try_fmt_vbi_out pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for raw VBI output

vidioc_try_fmt_sliced_vbi_cap pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for sliced VBI capture

vidioc_try_fmt_sliced_vbi_out pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for sliced VBI output

vidioc_try_fmt_vid_cap_mplane pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video capture in multiple plane mode

vidioc_try_fmt_vid_out_mplane pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video out in multiplane plane mode

vidioc_try_fmt_sdr_cap pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for Software Defined Radio capture

vidioc_try_fmt_sdr_out pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for Software Defined Radio output

vidioc_reqbufs pointer to the function that implements *VIDIOC_REQBUFS* ioctl

vidioc_querybuf pointer to the function that implements *VIDIOC_QUERYBUF* ioctl

vidioc_qbuf pointer to the function that implements *VIDIOC_QBUF* ioctl

vidioc_expbuf pointer to the function that implements *VIDIOC_EXPBUF* ioctl

vidioc_dqbuf pointer to the function that implements *VIDIOC_DQBUF* ioctl

vidioc_create_bufs pointer to the function that implements *VIDIOC_CREATE_BUFS* ioctl

vidioc_prepare_buf pointer to the function that implements *VIDIOC_PREPARE_BUF* ioctl

vidioc_overlay pointer to the function that implements *VIDIOC_OVERLAY* ioctl

vidioc_g_fbuf pointer to the function that implements *VIDIOC_G_FBUF* ioctl

vidioc_s_fbuf pointer to the function that implements *VIDIOC_S_FBUF* ioctl

vidioc_streamon pointer to the function that implements *VIDIOC_STREAMON* ioctl

vidioc_streamoff pointer to the function that implements *VIDIOC_STREAMOFF* ioctl

vidioc_g_std pointer to the function that implements *VIDIOC_G_STD* ioctl

vidioc_s_std pointer to the function that implements *VIDIOC_S_STD* ioctl

vidioc_querystd pointer to the function that implements *VIDIOC_QUERYSTD* ioctl

vidioc_enum_input pointer to the function that implements *VIDIOC_ENUM_INPUT* ioctl

vidioc_g_input pointer to the function that implements *VIDIOC_G_INPUT* ioctl

vidioc_s_input pointer to the function that implements *VIDIOC_S_INPUT* ioctl
vidioc_enum_output pointer to the function that implements *VIDIOC_ENUM_OUTPUT* ioctl
vidioc_g_output pointer to the function that implements *VIDIOC_G_OUTPUT* ioctl
vidioc_s_output pointer to the function that implements *VIDIOC_S_OUTPUT* ioctl
vidioc_queryctrl pointer to the function that implements *VIDIOC_QUERYCTRL* ioctl
vidioc_query_ext_ctrl pointer to the function that implements *VIDIOC_QUERY_EXT_CTRL* ioctl
vidioc_g_ctrl pointer to the function that implements *VIDIOC_G_CTRL* ioctl
vidioc_s_ctrl pointer to the function that implements *VIDIOC_S_CTRL* ioctl
vidioc_g_ext_ctrls pointer to the function that implements *VIDIOC_G_EXT_CTRLS* ioctl
vidioc_s_ext_ctrls pointer to the function that implements *VIDIOC_S_EXT_CTRLS* ioctl
vidioc_try_ext_ctrls pointer to the function that implements *VIDIOC_TRY_EXT_CTRLS* ioctl
vidioc_querymenu pointer to the function that implements *VIDIOC_QUERYMENU* ioctl
vidioc_enumaudio pointer to the function that implements *VIDIOC_ENUMAUDIO* ioctl
vidioc_g_audio pointer to the function that implements *VIDIOC_G_AUDIO* ioctl
vidioc_s_audio pointer to the function that implements *VIDIOC_S_AUDIO* ioctl
vidioc_enumaudout pointer to the function that implements *VIDIOC_ENUMAUDOUT* ioctl
vidioc_g_audout pointer to the function that implements *VIDIOC_G_AUDOUT* ioctl
vidioc_s_audout pointer to the function that implements *VIDIOC_S_AUDOUT* ioctl
vidioc_g_modulator pointer to the function that implements *VIDIOC_G_MODULATOR* ioctl
vidioc_s_modulator pointer to the function that implements *VIDIOC_S_MODULATOR* ioctl
vidioc_cropcap pointer to the function that implements *VIDIOC_CROPCAP* ioctl
vidioc_g_crop pointer to the function that implements *VIDIOC_G_CROP* ioctl
vidioc_s_crop pointer to the function that implements *VIDIOC_S_CROP* ioctl
vidioc_g_selection pointer to the function that implements *VIDIOC_G_SELECTION* ioctl
vidioc_s_selection pointer to the function that implements *VIDIOC_S_SELECTION* ioctl
vidioc_g_jpegcomp pointer to the function that implements *VIDIOC_G_JPEGCOMP* ioctl
vidioc_s_jpegcomp pointer to the function that implements *VIDIOC_S_JPEGCOMP* ioctl
vidioc_g_enc_index pointer to the function that implements *VIDIOC_G_ENC_INDEX* ioctl
vidioc_encoder_cmd pointer to the function that implements *VIDIOC_ENCODER_CMD* ioctl
vidioc_try_encoder_cmd pointer to the function that implements *VIDIOC_TRY_ENCODER_CMD* ioctl
vidioc_decoder_cmd pointer to the function that implements *VIDIOC_DECODER_CMD* ioctl
vidioc_try_decoder_cmd pointer to the function that implements *VIDIOC_TRY_DECODER_CMD* ioctl
vidioc_g_parm pointer to the function that implements *VIDIOC_G_PARM* ioctl
vidioc_s_parm pointer to the function that implements *VIDIOC_S_PARM* ioctl
vidioc_g_tuner pointer to the function that implements *VIDIOC_G_TUNER* ioctl
vidioc_s_tuner pointer to the function that implements *VIDIOC_S_TUNER* ioctl
vidioc_g_frequency pointer to the function that implements *VIDIOC_G_FREQUENCY* ioctl
vidioc_s_frequency pointer to the function that implements *VIDIOC_S_FREQUENCY* ioctl
vidioc_enum_freq_bands pointer to the function that implements *VIDIOC_ENUM_FREQ_BANDS* ioctl

vidioc_g_sliced_vbi_cap pointer to the function that implements *VIDIOC_G_SLICED_VBI_CAP* ioctl

vidioc_log_status pointer to the function that implements *VIDIOC_LOG_STATUS* ioctl

vidioc_s_hw_freq_seek pointer to the function that implements *VIDIOC_S_HW_FREQ_SEEK* ioctl

vidioc_g_register pointer to the function that implements *VIDIOC_DBG_G_REGISTER* ioctl

vidioc_s_register pointer to the function that implements *VIDIOC_DBG_S_REGISTER* ioctl

vidioc_g_chip_info pointer to the function that implements *VIDIOC_DBG_G_CHIP_INFO* ioctl

vidioc_enum_framesizes pointer to the function that implements *VIDIOC_ENUM_FRAMESIZES* ioctl

vidioc_enum_frameintervals pointer to the function that implements *VIDIOC_ENUM_FRAMEINTERVALS* ioctl

vidioc_s_dv_timings pointer to the function that implements *VIDIOC_S_DV_TIMINGS* ioctl

vidioc_g_dv_timings pointer to the function that implements *VIDIOC_G_DV_TIMINGS* ioctl

vidioc_query_dv_timings pointer to the function that implements *VIDIOC_QUERY_DV_TIMINGS* ioctl

vidioc_enum_dv_timings pointer to the function that implements *VIDIOC_ENUM_DV_TIMINGS* ioctl

vidioc_dv_timings_cap pointer to the function that implements *VIDIOC_DV_TIMINGS_CAP* ioctl

vidioc_g_edid pointer to the function that implements *VIDIOC_G_EDID* ioctl

vidioc_s_edid pointer to the function that implements *VIDIOC_S_EDID* ioctl

vidioc_subscribe_event pointer to the function that implements *VIDIOC_SUBSCRIBE_EVENT* ioctl

vidioc_unsubscribe_event pointer to the function that implements *VIDIOC_UNSUBSCRIBE_EVENT* ioctl

vidioc_default pointed used to allow other ioctls

const char * **v4l2_norm_to_name**(v4l2_std_id *id*)
Ancillary routine to analog TV standard name from its ID.

Parameters

v4l2_std_id id analog TV standard ID.

Return

returns a string with the name of the analog TV standard. If the standard is not found or if **id** points to multiple standard, it returns “Unknown”.

void **v4l2_video_std_frame_period**(int *id*, struct *v4l2_fract* * *frameperiod*)
Ancillary routine that fills a struct *v4l2_fract* pointer with the default framerate fraction.

Parameters

int id analog TV sdandard ID.

struct v4l2_fract * frameperiod struct *v4l2_fract* pointer to be filled

int **v4l2_video_std_construct**(struct *v4l2_standard* * *vs*, int *id*, const char * *name*)
Ancillary routine that fills in the fields of a *v4l2_standard* structure according to the **id** parameter.

Parameters

struct v4l2_standard * vs struct *v4l2_standard* pointer to be filled

int id analog TV sdandard ID.

const char * name name of the standard to be used

Description

Note:

This ancillary routine is obsolete. Shouldn't be used on newer drivers.

void **v4l_printk_ioctl**(const char * *prefix*, unsigned int *cmd*)
Ancillary routine that prints the ioctl in a human-readable format.

Parameters

const char * prefix prefix to be added at the ioctl prints.

unsigned int cmd ioctl name

Description

Note:

If prefix != NULL, then it will issue a `printk(KERN_DEBUG "``s: ", prefix)'' first.

struct mutex * **v4l2_ioctl_get_lock**(struct *video_device* * *vdev*, unsigned int *cmd*)
get the mutex (if any) that it is need to lock for a given command.

Parameters

struct video_device * vdev Pointer to struct *video_device*.

unsigned int cmd ioctl name.

Description

Note:

Internal use only. Should not be used outside V4L2 core.

long int **v4l2_compat_ioctl32**(struct file * *file*, unsigned int *cmd*, unsigned long *arg*)
32 Bits compatibility layer for 64 bits processors

Parameters

struct file * file Pointer to struct `file`.

unsigned int cmd ioctl name.

unsigned long arg ioctl argument.

v4l2_kioctl
Typedef: Typedef used to pass an ioctl handler.

Syntax

```
long v4l2_kioctl (struct file * file,unsigned int cmd,void * arg);
```

Parameters

struct file * file Pointer to struct `file`.

unsigned int cmd ioctl name.

void * arg ioctl argument.

long int **video_usercopy**(struct file * *file*, unsigned int *cmd*, unsigned long int *arg*, *v4l2_kioctl func*)
copies data from/to userspace memory when an ioctl is issued.

Parameters

struct file * file Pointer to struct file.

unsigned int cmd Ioctl name.

unsigned long int arg Ioctl argument.

v4l2_kioctl func function that will handle the ioctl

Description

Note:

This routine should be used only inside the V4L2 core.

long int **video_ioctl2**(struct file * *file*, unsigned int *cmd*, unsigned long int *arg*)
Handles a V4L2 ioctl.

Parameters

struct file * file Pointer to struct file.

unsigned int cmd Ioctl name.

unsigned long int arg Ioctl argument.

Description

Method used to hancle an ioctl. Should be used to fill the *v4l2_ioctl_ops.unlocked_ioctl* on all V4L2 drivers.

2.1.25 Hauppauge TV EEPROM functions and data structures

enum **tveeprom_audio_processor**
Specifies the type of audio processor used on a Hauppauge device.

Constants

TVEEPROM_AUDPROC_NONE No audio processor present

TVEEPROM_AUDPROC_INTERNAL The audio processor is internal to the video processor

TVEEPROM_AUDPROC_MSP The audio processor is a MSPXXXX device

TVEEPROM_AUDPROC_OTHER The audio processor is another device

struct **tveeprom**
Contains the fields parsed from Hauppauge eeproms

Definition

```
struct tveeprom {
  u32 has_radio;
  u32 has_ir;
  u32 has_MAC_address;
  u32 tuner_type;
  u32 tuner_formats;
  u32 tuner_hauppauge_model;
  u32 tuner2_type;
  u32 tuner2_formats;
  u32 tuner2_hauppauge_model;
  u32 audio_processor;
  u32 decoder_processor;
  u32 model;
  u32 revision;
  u32 serial_number;
  char rev_str[5];
```

```
  u8 MAC_address[ETH_ALEN];
};
```

Members

has_radio 1 if the device has radio; 0 otherwise.

has_ir If has_ir == 0, then it is unknown what the IR capabilities are. Otherwise: bit 0) 1 (= IR capabilities are known); bit 1) IR receiver present; bit 2) IR transmitter (blaster) present.

has_MAC_address 0: no MAC, 1: MAC present, 2: unknown.

tuner_type type of the tuner (TUNER_*, as defined at include/media/tuner.h).

tuner_formats Supported analog TV standards (V4L2_STD_*).

tuner_hauppauge_model Hauppauge's code for the device model number.

tuner2_type type of the second tuner (TUNER_*, as defined at include/media/tuner.h).

tuner2_formats Tuner 2 supported analog TV standards (V4L2_STD_*).

tuner2_hauppauge_model tuner 2 Hauppauge's code for the device model number.

audio_processor analog audio decoder, as defined by enum tveeprom_audio_processor.

decoder_processor Hauppauge's code for the decoder chipset. Unused by the drivers, as they probe the decoder based on the PCI or USB ID.

model Hauppauge's model number

revision Card revision number

serial_number Card's serial number

rev_str[5] Card revision converted to number

MAC_address[ETH_ALEN] MAC address for the network interface

void **tveeprom_hauppauge_analog**(struct i2c_client * *c*, struct *tveeprom* * *tvee*, unsigned char * *eeprom_data*)

Fill struct tveeprom using the contents of the eeprom previously filled at **eeprom_data** field.

Parameters

struct i2c_client * c I2C client struct

struct tveeprom * tvee Struct to where the eeprom parsed data will be filled;

unsigned char * eeprom_data Array with the contents of the eeprom_data. It should contain 256 bytes filled with the contents of the eeprom read from the Hauppauge device.

int **tveeprom_read**(struct i2c_client * *c*, unsigned char * *eedata*, int *len*)

Reads the contents of the eeprom found at the Hauppauge devices.

Parameters

struct i2c_client * c I2C client struct

unsigned char * eedata Array where the eeprom content will be stored.

int len Size of **eedata** array. If the eeprom content will be latter be parsed by *tveeprom_hauppauge_analog()*, len should be, at least, 256.

2.2 Digital TV (DVB) devices

2.3 Digital TV Common functions

unsigned int **intlog2**(u32 *value*)
computes log2 of a value; the result is shifted left by 24 bits

Parameters

u32 value The value (must be != 0)

Description

to use rational values you can use the following method:

intlog2(value) = intlog2(value * 2^x) - x * 2^24

Some usecase examples:

intlog2(8) will give 3 << 24 = 3 * 2^24

intlog2(9) will give 3 << 24 + ... = 3.16... * 2^24

intlog2(1.5) = intlog2(3) - 2^24 = 0.584... * 2^24

Return

log2(value) * 2^24

unsigned int **intlog10**(u32 *value*)
computes log10 of a value; the result is shifted left by 24 bits

Parameters

u32 value The value (must be != 0)

Description

to use rational values you can use the following method:

intlog10(value) = intlog10(value * 10^x) - x * 2^24

An usecase example:

intlog10(1000) will give 3 << 24 = 3 * 2^24

due to the implementation intlog10(1000) might be not exactly 3 * 2^24

look at intlog2 for similar examples

Return

log10(value) * 2^24

struct **dvb_adapter**
represents a Digital TV adapter using Linux DVB API

Definition

```
struct dvb_adapter {
  int num;
  struct list_head list_head;
  struct list_head device_list;
  const char * name;
  u8 proposed_mac[6];
  void * priv;
  struct device * device;
  struct module * module;
  int mfe_shared;
  struct dvb_device * mfe_dvbdev;
```

```
  struct mutex mfe_lock;
#if defined(CONFIG_MEDIA_CONTROLLER_DVB)
  struct media_device * mdev;
  struct media_entity * conn;
  struct media_pad * conn_pads;
#endif
};
```

Members

num Number of the adapter

list_head List with the DVB adapters

device_list List with the DVB devices

name Name of the adapter

proposed_mac[6] proposed MAC address for the adapter

priv private data

device pointer to struct device

module pointer to struct module

mfe_shared mfe shared: indicates mutually exclusive frontends Thie usage of this flag is currently deprecated

mfe_dvbdev Frontend device in use, in the case of MFE

mfe_lock Lock to prevent using the other frontends when MFE is used.

mdev pointer to struct media_device, used when the media controller is used.

conn RF connector. Used only if the device has no separate tuner.

conn_pads pointer to struct media_pad associated with **conn**;

struct **dvb_device**
represents a DVB device node

Definition

```
struct dvb_device {
  struct list_head list_head;
  const struct file_operations * fops;
  struct dvb_adapter * adapter;
  int type;
  int minor;
  u32 id;
  int readers;
  int writers;
  int users;
  wait_queue_head_t wait_queue;
  int (* kernel_ioctl) (struct file *file, unsigned int cmd, void *arg);
#if defined(CONFIG_MEDIA_CONTROLLER_DVB)
  const char * name;
  struct media_intf_devnode * intf_devnode;
  unsigned tsout_num_entities;
  struct media_entity * entity;
  struct media_entity * tsout_entity;
  struct media_pad * pads;
  struct media_pad * tsout_pads;
#endif
  void * priv;
};
```

Members

list_head List head with all DVB devices

fops pointer to struct file_operations

adapter pointer to the adapter that holds this device node

type type of the device: DVB_DEVICE_SEC, DVB_DEVICE_FRONTEND, DVB_DEVICE_DEMUX, DVB_DEVICE_DVR, DVB_DEVICE_CA, DVB_DEVICE_NET

minor devnode minor number. Major number is always DVB_MAJOR.

id device ID number, inside the adapter

readers Initialized by the caller. Each call to *open()* in Read Only mode decreases this counter by one.

writers Initialized by the caller. Each call to *open()* in Read/Write mode decreases this counter by one.

users Initialized by the caller. Each call to *open()* in any mode decreases this counter by one.

wait_queue wait queue, used to wait for certain events inside one of the DVB API callers

kernel_ioctl callback function used to handle ioctl calls from userspace.

name Name to be used for the device at the Media Controller

intf_devnode Pointer to media_intf_devnode. Used by the dvbdev core to store the MC device node interface

tsout_num_entities Number of Transport Stream output entities

entity pointer to struct media_entity associated with the device node

tsout_entity array with MC entities associated to each TS output node

pads pointer to struct media_pad associated with **entity**;

tsout_pads array with the source pads for each **tsout_entity**

priv private data

Description

This structure is used by the DVB core (frontend, CA, net, demux) in order to create the device nodes. Usually, driver should not initialize this struct diretly.

int **dvb_register_adapter**(struct *dvb_adapter* * *adap*, const char * *name*, struct module * *module*, struct device * *device*, short * *adapter_nums*)
Registers a new DVB adapter

Parameters

struct dvb_adapter * adap pointer to struct dvb_adapter

const char * name Adapter's name

struct module * module initialized with THIS_MODULE at the caller

struct device * device pointer to struct device that corresponds to the device driver

short * adapter_nums Array with a list of the numbers for **dvb_register_adapter**; to select among them. Typically, initialized with: DVB_DEFINE_MOD_OPT_ADAPTER_NR(adapter_nums)

int **dvb_unregister_adapter**(struct *dvb_adapter* * *adap*)
Unregisters a DVB adapter

Parameters

struct dvb_adapter * adap pointer to struct dvb_adapter

int **dvb_register_device**(struct *dvb_adapter* * *adap*, struct *dvb_device* ** *pdvbdev*, const struct *dvb_device* * *template*, void * *priv*, int *type*, int *demux_sink_pads*)
Registers a new DVB device

Parameters

struct dvb_adapter * adap pointer to struct dvb_adapter

struct dvb_device ** pdvbdev pointer to the place where the new struct dvb_device will be stored

const struct dvb_device * template Template used to create pdvbdev;

void * priv private data

int type type of the device: DVB_DEVICE_SEC, DVB_DEVICE_FRONTEND, DVB_DEVICE_DEMUX, DVB_DEVICE_DVR, DVB_DEVICE_CA, DVB_DEVICE_NET

int demux_sink_pads Number of demux outputs, to be used to create the TS outputs via the Media Controller.

void **dvb_remove_device**(struct *dvb_device* * *dvbdev*)
Remove a registered DVB device

Parameters

struct dvb_device * dvbdev pointer to struct dvb_device

Description

This does not free memory. To do that, call *dvb_free_device()*.

void **dvb_free_device**(struct *dvb_device* * *dvbdev*)
Free memory occupied by a DVB device.

Parameters

struct dvb_device * dvbdev pointer to struct dvb_device

Description

Call *dvb_unregister_device()* before calling this function.

void **dvb_unregister_device**(struct *dvb_device* * *dvbdev*)
Unregisters a DVB device

Parameters

struct dvb_device * dvbdev pointer to struct dvb_device

Description

This is a combination of *dvb_remove_device()* and *dvb_free_device()*. Using this function is usually a mistake, and is often an indicator for a use-after-free bug (when a userspace process keeps a file handle to a detached device).

int **dvb_create_media_graph**(struct *dvb_adapter* * *adap*, bool *create_rf_connector*)
Creates media graph for the Digital TV part of the device.

Parameters

struct dvb_adapter * adap pointer to struct dvb_adapter

bool create_rf_connector if true, it creates the RF connector too

Description

This function checks all DVB-related functions at the media controller entities and creates the needed links for the media graph. It is capable of working with multiple tuners or multiple frontends, but it won't create links if the device has multiple tuners and multiple frontends or if the device has multiple muxes. In such case, the caller driver should manually create the remaining links.

2.4 Digital TV Ring buffer

Those routines implement ring buffers used to handle digital TV data and copy it from/to userspace.

Note:

1. *For performance reasons read and write routines don't check buffer sizes and/or number of bytes free/available. This has to be done before these routines are called. For example:*

```
/* write @buflen: bytes */
free = dvb_ringbuffer_free(rbuf);
if (free >= buflen)
        count = dvb_ringbuffer_write(rbuf, buffer, buflen);
else
        /* do something */

/* read min. 1000, max. @bufsize: bytes */
avail = dvb_ringbuffer_avail(rbuf);
if (avail >= 1000)
        count = dvb_ringbuffer_read(rbuf, buffer, min(avail, bufsize));
else
        /* do something */
```

2. *If there is exactly one reader and one writer, there is no need to lock read or write operations. Two or more readers must be locked against each other. Flushing the buffer counts as a read operation. Resetting the buffer counts as a read and write operation. Two or more writers must be locked against each other.*

struct **dvb_ringbuffer**
Describes a ring buffer used at DVB framework

Definition

```
struct dvb_ringbuffer {
  u8 * data;
  ssize_t size;
  ssize_t pread;
  ssize_t pwrite;
  int error;
  wait_queue_head_t queue;
  spinlock_t lock;
};
```

Members

data Area were the ringbuffer data is written

size size of the ringbuffer

pread next position to read

pwrite next position to write

error used by ringbuffer clients to indicate that an error happened.

queue Wait queue used by ringbuffer clients to indicate when buffer was filled

lock Spinlock used to protect the ringbuffer

void **dvb_ringbuffer_init**(struct *dvb_ringbuffer* * *rbuf*, void * *data*, size_t *len*)
initialize ring buffer, lock and queue

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

void * data pointer to the buffer where the data will be stored

size_t len bytes from ring buffer into **buf**

int **dvb_ringbuffer_empty**(struct *dvb_ringbuffer * rbuf*)
test whether buffer is empty

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

ssize_t **dvb_ringbuffer_free**(struct *dvb_ringbuffer * rbuf*)
returns the number of free bytes in the buffer

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

Return

number of free bytes in the buffer

ssize_t **dvb_ringbuffer_avail**(struct *dvb_ringbuffer * rbuf*)
returns the number of bytes waiting in the buffer

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

Return

number of bytes waiting in the buffer

void **dvb_ringbuffer_reset**(struct *dvb_ringbuffer * rbuf*)
resets the ringbuffer to initial state

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

Description

Resets the read and write pointers to zero and flush the buffer.

This counts as a read and write operation

void **dvb_ringbuffer_flush**(struct *dvb_ringbuffer * rbuf*)
flush buffer

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

void **dvb_ringbuffer_flush_spinlock_wakeup**(struct *dvb_ringbuffer * rbuf*)
flush buffer protected by spinlock and wake-up waiting task(s)

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

DVB_RINGBUFFER_PEEK(*rbuf*, *offs*)
peek at byte **offs** in the buffer

Parameters

rbuf pointer to struct dvb_ringbuffer

offs offset inside the ringbuffer

DVB_RINGBUFFER_SKIP(*rbuf*, *num*)
advance read ptr by **num** bytes

Parameters

rbuf pointer to struct dvb_ringbuffer

num number of bytes to advance

ssize_t **dvb_ringbuffer_read_user**(struct *dvb_ringbuffer* * *rbuf*, u8 __user * *buf*, size_t *len*)
Reads a buffer into an user pointer

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

u8 __user * buf pointer to the buffer where the data will be stored

size_t len bytes from ring buffer into **buf**

Description

This variant assumes that the buffer is a memory at the userspace. So, it will internally call `copy_to_user()`.

Return

number of bytes transferred or -EFAULT

void **dvb_ringbuffer_read**(struct *dvb_ringbuffer* * *rbuf*, u8 * *buf*, size_t *len*)
Reads a buffer into a pointer

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

u8 * buf pointer to the buffer where the data will be stored

size_t len bytes from ring buffer into **buf**

Description

This variant assumes that the buffer is a memory at the Kernel space

Return

number of bytes transferred or -EFAULT

DVB_RINGBUFFER_WRITE_BYTE(*rbuf*, *byte*)
write single byte to ring buffer

Parameters

rbuf pointer to struct dvb_ringbuffer

byte byte to write

ssize_t **dvb_ringbuffer_write**(struct *dvb_ringbuffer* * *rbuf*, const u8 * *buf*, size_t *len*)
Writes a buffer into the ringbuffer

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

const u8 * buf pointer to the buffer where the data will be read

size_t len bytes from ring buffer into **buf**

Description

This variant assumes that the buffer is a memory at the Kernel space

Return

number of bytes transferred or -EFAULT

ssize_t **dvb_ringbuffer_write_user**(struct *dvb_ringbuffer* * *rbuf*, const u8 __user * *buf*, size_t *len*)
Writes a buffer received via an user pointer

Parameters

struct dvb_ringbuffer * rbuf pointer to struct dvb_ringbuffer

const u8 __user * buf pointer to the buffer where the data will be read

size_t len bytes from ring buffer into **buf**

Description

This variant assumes that the buffer is a memory at the userspace. So, it will internally call `copy_from_user()`.

Return

number of bytes transferred or -EFAULT

ssize_t **dvb_ringbuffer_pkt_write**(struct *dvb_ringbuffer* * *rbuf*, u8 * *buf*, size_t *len*)

Write a packet into the ringbuffer.

Parameters

struct dvb_ringbuffer * rbuf Ringbuffer to write to.

u8 * buf Buffer to write.

size_t len Length of buffer (currently limited to 65535 bytes max).

Return

Number of bytes written, or -EFAULT, -ENOMEM, -EVINAL.

ssize_t **dvb_ringbuffer_pkt_read_user**(struct *dvb_ringbuffer* * *rbuf*, size_t *idx*, int *offset*, u8 __user * *buf*, size_t *len*)

Read from a packet in the ringbuffer.

Parameters

struct dvb_ringbuffer * rbuf Ringbuffer concerned.

size_t idx Packet index as returned by *dvb_ringbuffer_pkt_next()*.

int offset Offset into packet to read from.

u8 __user * buf Destination buffer for data.

size_t len Size of destination buffer.

Return

Number of bytes read, or -EFAULT.

Note:

unlike `dvb_ringbuffer_read()`*, this does* ***NOT*** *update the read pointer in the ringbuffer. You must use* `dvb_ringbuffer_pkt_dispose()` *to mark a packet as no longer required.*

ssize_t **dvb_ringbuffer_pkt_read**(struct *dvb_ringbuffer* * *rbuf*, size_t *idx*, int *offset*, u8 * *buf*, size_t *len*)

Read from a packet in the ringbuffer.

Parameters

struct dvb_ringbuffer * rbuf Ringbuffer concerned.

size_t idx Packet index as returned by *dvb_ringbuffer_pkt_next()*.

int offset Offset into packet to read from.

u8 * buf Destination buffer for data.

size_t len Size of destination buffer.

Note

unlike *dvb_ringbuffer_read_user()*, this DOES update the read pointer in the ringbuffer.

Return

Number of bytes read, or -EFAULT.

void **dvb_ringbuffer_pkt_dispose**(struct *dvb_ringbuffer* * *rbuf*, size_t *idx*)
Dispose of a packet in the ring buffer.

Parameters

struct dvb_ringbuffer * rbuf Ring buffer concerned.

size_t idx Packet index as returned by *dvb_ringbuffer_pkt_next()*.

ssize_t **dvb_ringbuffer_pkt_next**(struct *dvb_ringbuffer* * *rbuf*, size_t *idx*, size_t * *pktlen*)
Get the index of the next packet in a ringbuffer.

Parameters

struct dvb_ringbuffer * rbuf Ringbuffer concerned.

size_t idx Previous packet index, or -1 to return the first packet index.

size_t * pktlen On success, will be updated to contain the length of the packet in bytes. returns Packet index (if >=0), or -1 if no packets available.

2.5 Digital TV Frontend kABI

2.5.1 Digital TV Frontend

The Digital TV Frontend kABI defines a driver-internal interface for registering low-level, hardware specific driver to a hardware independent frontend layer. It is only of interest for Digital TV device driver writers. The header file for this API is named dvb_frontend.h and located in drivers/media/dvb-core.

Before using the Digital TV frontend core, the bridge driver should attach the frontend demod, tuner and SEC devices and call *dvb_register_frontend()*, in order to register the new frontend at the subsystem. At device detach/removal, the bridge driver should call *dvb_unregister_frontend()* to remove the frontend from the core and then *dvb_frontend_detach()* to free the memory allocated by the frontend drivers.

The drivers should also call *dvb_frontend_suspend()* as part of their handler for the device_driver.suspend(), and *dvb_frontend_resume()* as part of their handler for device_driver.resume().

A few other optional functions are provided to handle some special cases.

struct **dvb_frontend_tune_settings**
parameters to adjust frontend tuning

Definition

```
struct dvb_frontend_tune_settings {
  int min_delay_ms;
  int step_size;
  int max_drift;
};
```

Members

min_delay_ms minimum delay for tuning, in ms

step_size step size between two consecutive frequencies

max_drift maximum drift

NOTE

step_size is in Hz, for terrestrial/cable or kHz for satellite

struct **dvb_tuner_info**
Frontend name and min/max ranges/bandwidths

Definition

```
struct dvb_tuner_info {
  char name[128];
  u32 frequency_min;
  u32 frequency_max;
  u32 frequency_step;
  u32 bandwidth_min;
  u32 bandwidth_max;
  u32 bandwidth_step;
};
```

Members

name[128] name of the Frontend

frequency_min minimal frequency supported

frequency_max maximum frequency supported

frequency_step frequency step

bandwidth_min minimal frontend bandwidth supported

bandwidth_max maximum frontend bandwidth supported

bandwidth_step frontend bandwidth step

NOTE

frequency parameters are in Hz, for terrestrial/cable or kHz for satellite.

struct **analog_parameters**
Parameters to tune into an analog/radio channel

Definition

```
struct analog_parameters {
  unsigned int frequency;
  unsigned int mode;
  unsigned int audmode;
  u64 std;
};
```

Members

frequency Frequency used by analog TV tuner (either in 62.5 kHz step, for TV, or 62.5 Hz for radio)

mode Tuner mode, as defined on enum v4l2_tuner_type

audmode Audio mode as defined for the rxsubchans field at videodev2.h, e. g. V4L2_TUNER_MODE_*

std TV standard bitmap as defined at videodev2.h, e. g. V4L2_STD_*

Description

Hybrid tuners should be supported by both V4L2 and DVB APIs. This struct contains the data that are used by the V4L2 side. To avoid dependencies from V4L2 headers, all enums here are declared as integers.

enum **dvbfe_algo**
defines the algorithm used to tune into a channel

Constants

DVBFE_ALGO_HW Hardware Algorithm - Devices that support this algorithm do everything in hardware and no software support is needed to handle them. Requesting these devices to LOCK is the only thing required, device is supposed to do everything in the hardware.

DVBFE_ALGO_SW Software Algorithm - These are dumb devices, that require software to do everything

DVBFE_ALGO_CUSTOM Customizable Agorithm - Devices having this algorithm can be customized to have specific algorithms in the frontend driver, rather than simply doing a software zig-zag. In this case the zigzag maybe hardware assisted or it maybe completely done in hardware. In all cases, usage of this algorithm, in conjunction with the search and track callbacks, utilizes the driver specific algorithm.

DVBFE_ALGO_RECOVERY Recovery Algorithm - These devices have AUTO recovery capabilities from LOCK failure

enum **dvbfe_search**
search callback possible return status

Constants

DVBFE_ALGO_SEARCH_SUCCESS The frontend search algorithm completed and returned successfully

DVBFE_ALGO_SEARCH_ASLEEP The frontend search algorithm is sleeping

DVBFE_ALGO_SEARCH_FAILED The frontend search for a signal failed

DVBFE_ALGO_SEARCH_INVALID The frontend search algorith was probably supplied with invalid parameters and the search is an invalid one

DVBFE_ALGO_SEARCH_AGAIN The frontend search algorithm was requested to search again

DVBFE_ALGO_SEARCH_ERROR The frontend search algorithm failed due to some error

struct **dvb_tuner_ops**
Tuner information and callbacks

Definition

```
struct dvb_tuner_ops {
  struct dvb_tuner_info info;
  void (* release) (struct dvb_frontend *fe);
  int (* init) (struct dvb_frontend *fe);
  int (* sleep) (struct dvb_frontend *fe);
  int (* suspend) (struct dvb_frontend *fe);
  int (* resume) (struct dvb_frontend *fe);
  int (* set_params) (struct dvb_frontend *fe);
  int (* set_analog_params) (struct dvb_frontend *fe, struct analog_parameters *p);
  int (* set_config) (struct dvb_frontend *fe, void *priv_cfg);
  int (* get_frequency) (struct dvb_frontend *fe, u32 *frequency);
  int (* get_bandwidth) (struct dvb_frontend *fe, u32 *bandwidth);
  int (* get_if_frequency) (struct dvb_frontend *fe, u32 *frequency);
#define TUNER_STATUS_LOCKED 1
#define TUNER_STATUS_STEREO 2
  int (* get_status) (struct dvb_frontend *fe, u32 *status);
  int (* get_rf_strength) (struct dvb_frontend *fe, u16 *strength);
  int (* get_afc) (struct dvb_frontend *fe, s32 *afc);
  int (* calc_regs) (struct dvb_frontend *fe, u8 *buf, int buf_len);
  int (* set_frequency) (struct dvb_frontend *fe, u32 frequency);
  int (* set_bandwidth) (struct dvb_frontend *fe, u32 bandwidth);
};
```

Members

info embedded struct dvb_tuner_info with tuner properties

release callback function called when frontend is dettached. drivers should free any allocated memory.

init callback function used to initialize the tuner device.

sleep callback function used to put the tuner to sleep.

suspend callback function used to inform that the Kernel will suspend.

resume callback function used to inform that the Kernel is resuming from suspend.

set_params callback function used to inform the tuner to tune into a digital TV channel. The properties to be used are stored at **dvb_frontend**.dtv_property_cache;. The tuner demod can change the parameters to reflect the changes needed for the channel to be tuned, and update statistics. This is the recommended way to set the tuner parameters and should be used on newer drivers.

set_analog_params callback function used to tune into an analog TV channel on hybrid tuners. It passes **analog_parameters**; to the driver.

set_config callback function used to send some tuner-specific parameters.

get_frequency get the actual tuned frequency

get_bandwidth get the bandwitdh used by the low pass filters

get_if_frequency get the Intermediate Frequency, in Hz. For baseband, should return 0.

get_status returns the frontend lock status

get_rf_strength returns the RF signal strengh. Used mostly to support analog TV and radio. Digital TV should report, instead, via DVBv5 API (**dvb_frontend**.dtv_property_cache;).

get_afc Used only by analog TV core. Reports the frequency drift due to AFC.

calc_regs callback function used to pass register data settings for simple tuners. Shouldn't be used on newer drivers.

set_frequency Set a new frequency. Shouldn't be used on newer drivers.

set_bandwidth Set a new frequency. Shouldn't be used on newer drivers.

NOTE

frequencies used on get_frequency and set_frequency are in Hz for terrestrial/cable or kHz for satellite.

struct **analog_demod_info**
Information struct for analog TV part of the demod

Definition

```
struct analog_demod_info {
  char * name;
};
```

Members

name Name of the analog TV demodulator

struct **analog_demod_ops**
Demodulation information and callbacks for analog TV and radio

Definition

```
struct analog_demod_ops {
  struct analog_demod_info info;
  void (* set_params) (struct dvb_frontend *fe,struct analog_parameters *params);
  int (* has_signal) (struct dvb_frontend *fe, u16 *signal);
  int (* get_afc) (struct dvb_frontend *fe, s32 *afc);
  void (* tuner_status) (struct dvb_frontend *fe);
  void (* standby) (struct dvb_frontend *fe);
  void (* release) (struct dvb_frontend *fe);
  int (* i2c_gate_ctrl) (struct dvb_frontend *fe, int enable);
  int (* set_config) (struct dvb_frontend *fe, void *priv_cfg);
};
```

Members

info pointer to struct analog_demod_info

set_params callback function used to inform the demod to set the demodulator parameters needed to decode an analog or radio channel. The properties are passed via struct **analog_params**;.

has_signal returns 0xffff if has signal, or 0 if it doesn't.

get_afc Used only by analog TV core. Reports the frequency drift due to AFC.

tuner_status callback function that returns tuner status bits, e. g. TUNER_STATUS_LOCKED and TUNER_STATUS_STEREO.

standby set the tuner to standby mode.

release callback function called when frontend is dettached. drivers should free any allocated memory.

i2c_gate_ctrl controls the I2C gate. Newer drivers should use I2C mux support instead.

set_config callback function used to send some tuner-specific parameters.

struct **dvb_frontend_ops**
Demodulation information and callbacks for ditialt TV

Definition

```
struct dvb_frontend_ops {
  struct dvb_frontend_info info;
  u8 delsys[MAX_DELSYS];
  void (* detach) (struct dvb_frontend *fe);
  void (* release) (struct dvb_frontend* fe);
  void (* release_sec) (struct dvb_frontend* fe);
  int (* init) (struct dvb_frontend* fe);
  int (* sleep) (struct dvb_frontend* fe);
  int (* write) (struct dvb_frontend* fe, const u8 buf[], int len);
  int (* tune) (struct dvb_frontend* fe,bool re_tune,unsigned int mode_flags,unsigned int␣
 ↪*delay,enum fe_status *status);
  enum dvbfe_algo (* get_frontend_algo) (struct dvb_frontend *fe);
  int (* set_frontend) (struct dvb_frontend *fe);
  int (* get_tune_settings) (struct dvb_frontend* fe, struct dvb_frontend_tune_settings*␣
 ↪settings);
  int (* get_frontend) (struct dvb_frontend *fe,struct dtv_frontend_properties *props);
  int (* read_status) (struct dvb_frontend *fe, enum fe_status *status);
  int (* read_ber) (struct dvb_frontend* fe, u32* ber);
  int (* read_signal_strength) (struct dvb_frontend* fe, u16* strength);
  int (* read_snr) (struct dvb_frontend* fe, u16* snr);
  int (* read_ucblocks) (struct dvb_frontend* fe, u32* ucblocks);
  int (* diseqc_reset_overload) (struct dvb_frontend* fe);
  int (* diseqc_send_master_cmd) (struct dvb_frontend* fe, struct dvb_diseqc_master_cmd* cmd);
  int (* diseqc_recv_slave_reply) (struct dvb_frontend* fe, struct dvb_diseqc_slave_reply*␣
 ↪reply);
  int (* diseqc_send_burst) (struct dvb_frontend *fe,enum fe_sec_mini_cmd minicmd);
  int (* set_tone) (struct dvb_frontend *fe, enum fe_sec_tone_mode tone);
  int (* set_voltage) (struct dvb_frontend *fe,enum fe_sec_voltage voltage);
  int (* enable_high_lnb_voltage) (struct dvb_frontend* fe, long arg);
  int (* dishnetwork_send_legacy_command) (struct dvb_frontend* fe, unsigned long cmd);
  int (* i2c_gate_ctrl) (struct dvb_frontend* fe, int enable);
  int (* ts_bus_ctrl) (struct dvb_frontend* fe, int acquire);
  int (* set_lna) (struct dvb_frontend *);
  enum dvbfe_search (* search) (struct dvb_frontend *fe);
  struct dvb_tuner_ops tuner_ops;
  struct analog_demod_ops analog_ops;
  int (* set_property) (struct dvb_frontend* fe, struct dtv_property* tvp);
  int (* get_property) (struct dvb_frontend* fe, struct dtv_property* tvp);
};
```

Members

info embedded struct dvb_tuner_info with tuner properties

delsys[MAX_DELSYS] Delivery systems supported by the frontend

detach callback function called when frontend is detached. drivers should clean up, but not yet free the struct dvb_frontend allocation.

release callback function called when frontend is ready to be freed. drivers should free any allocated memory.

release_sec callback function requesting that the Satelite Equipment Control (SEC) driver to release and free any memory allocated by the driver.

init callback function used to initialize the tuner device.

sleep callback function used to put the tuner to sleep.

write callback function used by some demod legacy drivers to allow other drivers to write data into their registers. Should not be used on new drivers.

tune callback function used by demod drivers that use **DVBFE_ALGO_HW**; to tune into a frequency.

get_frontend_algo returns the desired hardware algorithm.

set_frontend callback function used to inform the demod to set the parameters for demodulating a digital TV channel. The properties to be used are stored at **dvb_frontend**.dtv_property_cache;. The demod can change the parameters to reflect the changes needed for the channel to be decoded, and update statistics.

get_tune_settings callback function

get_frontend callback function used to inform the parameters actuall in use. The properties to be used are stored at **dvb_frontend**.dtv_property_cache; and update statistics. Please notice that it should not return an error code if the statistics are not available because the demog is not locked.

read_status returns the locking status of the frontend.

read_ber legacy callback function to return the bit error rate. Newer drivers should provide such info via DVBv5 API, e. g. **set_frontend**;/**get_frontend**;, implementing this callback only if DVBv3 API compatibility is wanted.

read_signal_strength legacy callback function to return the signal strength. Newer drivers should provide such info via DVBv5 API, e. g. **set_frontend**;/**get_frontend**;, implementing this callback only if DVBv3 API compatibility is wanted.

read_snr legacy callback function to return the Signal/Noise rate. Newer drivers should provide such info via DVBv5 API, e. g. **set_frontend**;/**get_frontend**;, implementing this callback only if DVBv3 API compatibility is wanted.

read_ucblocks legacy callback function to return the Uncorrected Error Blocks. Newer drivers should provide such info via DVBv5 API, e. g. **set_frontend**;/**get_frontend**;, implementing this callback only if DVBv3 API compatibility is wanted.

diseqc_reset_overload callback function to implement the FE_DISEQC_RESET_OVERLOAD ioctl (only Satellite)

diseqc_send_master_cmd callback function to implement the FE_DISEQC_SEND_MASTER_CMD ioctl (only Satellite).

diseqc_recv_slave_reply callback function to implement the FE_DISEQC_RECV_SLAVE_REPLY ioctl (only Satellite)

diseqc_send_burst callback function to implement the FE_DISEQC_SEND_BURST ioctl (only Satellite).

set_tone callback function to implement the FE_SET_TONE ioctl (only Satellite).

set_voltage callback function to implement the FE_SET_VOLTAGE ioctl (only Satellite).

enable_high_lnb_voltage callback function to implement the FE_ENABLE_HIGH_LNB_VOLTAGE ioctl (only Satellite).

dishnetwork_send_legacy_command callback function to implement the FE_DISHNETWORK_SEND_LEGACY_CMD ioctl (only Satellite). Drivers should not use this, except when the DVB core emulation fails to provide proper support (e.g. if **set_voltage** takes more than 8ms to work), and when backward compatibility with this legacy API is required.

i2c_gate_ctrl controls the I2C gate. Newer drivers should use I2C mux support instead.

ts_bus_ctrl callback function used to take control of the TS bus.

set_lna callback function to power on/off/auto the LNA.

search callback function used on some custom algo search algos.

tuner_ops pointer to struct dvb_tuner_ops

analog_ops pointer to struct analog_demod_ops

set_property callback function to allow the frontend to validade incoming properties. Should not be used on new drivers.

get_property callback function to allow the frontend to override outcoming properties. Should not be used on new drivers.

struct **dtv_frontend_properties**
contains a list of properties that are specific to a digital TV standard.

Definition

```
struct dtv_frontend_properties {
  u32 frequency;
  enum fe_modulation modulation;
  enum fe_sec_voltage voltage;
  enum fe_sec_tone_mode sectone;
  enum fe_spectral_inversion inversion;
  enum fe_code_rate fec_inner;
  enum fe_transmit_mode transmission_mode;
  u32 bandwidth_hz;
  enum fe_guard_interval guard_interval;
  enum fe_hierarchy hierarchy;
  u32 symbol_rate;
  enum fe_code_rate code_rate_HP;
  enum fe_code_rate code_rate_LP;
  enum fe_pilot pilot;
  enum fe_rolloff rolloff;
  enum fe_delivery_system delivery_system;
  enum fe_interleaving interleaving;
  u8 isdbt_partial_reception;
  u8 isdbt_sb_mode;
  u8 isdbt_sb_subchannel;
  u32 isdbt_sb_segment_idx;
  u32 isdbt_sb_segment_count;
  u8 isdbt_layer_enabled;
  struct layer[3];
  u32 stream_id;
  u8 atscmh_fic_ver;
  u8 atscmh_parade_id;
  u8 atscmh_nog;
  u8 atscmh_tnog;
  u8 atscmh_sgn;
  u8 atscmh_prc;
  u8 atscmh_rs_frame_mode;
  u8 atscmh_rs_frame_ensemble;
  u8 atscmh_rs_code_mode_pri;
  u8 atscmh_rs_code_mode_sec;
  u8 atscmh_sccc_block_mode;
  u8 atscmh_sccc_code_mode_a;
```

```
  u8 atscmh_sccc_code_mode_b;
  u8 atscmh_sccc_code_mode_c;
  u8 atscmh_sccc_code_mode_d;
  u32 lna;
  struct dtv_fe_stats strength;
  struct dtv_fe_stats cnr;
  struct dtv_fe_stats pre_bit_error;
  struct dtv_fe_stats pre_bit_count;
  struct dtv_fe_stats post_bit_error;
  struct dtv_fe_stats post_bit_count;
  struct dtv_fe_stats block_error;
  struct dtv_fe_stats block_count;
};
```

Members

frequency frequency in Hz for terrestrial/cable or in kHz for Satellite

modulation Frontend modulation type

voltage SEC voltage (only Satellite)

sectone SEC tone mode (only Satellite)

inversion Spectral inversion

fec_inner Forward error correction inner Code Rate

transmission_mode Transmission Mode

bandwidth_hz Bandwidth, in Hz. A zero value means that userspace wants to autodetect.

guard_interval Guard Interval

hierarchy Hierarchy

symbol_rate Symbol Rate

code_rate_HP high priority stream code rate

code_rate_LP low priority stream code rate

pilot Enable/disable/autodetect pilot tones

rolloff Rolloff factor (alpha)

delivery_system FE delivery system (e. g. digital TV standard)

interleaving interleaving

isdbt_partial_reception ISDB-T partial reception (only ISDB standard)

isdbt_sb_mode ISDB-T Sound Broadcast (SB) mode (only ISDB standard)

isdbt_sb_subchannel ISDB-T SB subchannel (only ISDB standard)

isdbt_sb_segment_idx ISDB-T SB segment index (only ISDB standard)

isdbt_sb_segment_count ISDB-T SB segment count (only ISDB standard)

isdbt_layer_enabled ISDB Layer enabled (only ISDB standard)

layer[3] per layer interleaving.

stream_id If different than zero, enable substream filtering, if hardware supports (DVB-S2 and DVB-T2).

atscmh_fic_ver Version number of the FIC (Fast Information Channel) signaling data (only ATSC-M/H)

atscmh_parade_id Parade identification number (only ATSC-M/H)

atscmh_nog Number of MH groups per MH subframe for a designated parade (only ATSC-M/H)

atscmh_tnog Total number of MH groups including all MH groups belonging to all MH parades in one MH subframe (only ATSC-M/H)

atscmh_sgn Start group number (only ATSC-M/H)

atscmh_prc Parade repetition cycle (only ATSC-M/H)

atscmh_rs_frame_mode Reed Solomon (RS) frame mode (only ATSC-M/H)

atscmh_rs_frame_ensemble RS frame ensemble (only ATSC-M/H)

atscmh_rs_code_mode_pri RS code mode pri (only ATSC-M/H)

atscmh_rs_code_mode_sec RS code mode sec (only ATSC-M/H)

atscmh_sccc_block_mode Series Concatenated Convolutional Code (SCCC) Block Mode (only ATSC-M/H)

atscmh_sccc_code_mode_a SCCC code mode A (only ATSC-M/H)

atscmh_sccc_code_mode_b SCCC code mode B (only ATSC-M/H)

atscmh_sccc_code_mode_c SCCC code mode C (only ATSC-M/H)

atscmh_sccc_code_mode_d SCCC code mode D (only ATSC-M/H)

lna Power ON/OFF/AUTO the Linear Now-noise Amplifier (LNA)

strength DVBv5 API statistics: Signal Strength

cnr DVBv5 API statistics: Signal to Noise ratio of the (main) carrier

pre_bit_error DVBv5 API statistics: pre-Viterbi bit error count

pre_bit_count DVBv5 API statistics: pre-Viterbi bit count

post_bit_error DVBv5 API statistics: post-Viterbi bit error count

post_bit_count DVBv5 API statistics: post-Viterbi bit count

block_error DVBv5 API statistics: block error count

block_count DVBv5 API statistics: block count

NOTE

derivated statistics like Uncorrected Error blocks (UCE) are calculated on userspace.

Only a subset of the properties are needed for a given delivery system. For more info, consult the media_api.html with the documentation of the Userspace API.

struct **dvb_frontend**
Frontend structure to be used on drivers.

Definition

```
struct dvb_frontend {
  struct dvb_frontend_ops ops;
  struct dvb_adapter * dvb;
  void * demodulator_priv;
  void * tuner_priv;
  void * frontend_priv;
  void * sec_priv;
  void * analog_demod_priv;
  struct dtv_frontend_properties dtv_property_cache;
#define DVB_FRONTEND_COMPONENT_TUNER 0
#define DVB_FRONTEND_COMPONENT_DEMOD 1
  int (* callback) (void *adapter_priv, int component, int cmd, int arg);
  int id;
  unsigned int exit;
};
```

Members

ops embedded struct dvb_frontend_ops

dvb pointer to struct dvb_adapter

demodulator_priv demod private data

tuner_priv tuner private data

frontend_priv frontend private data

sec_priv SEC private data

analog_demod_priv Analog demod private data

dtv_property_cache embedded struct dtv_frontend_properties

callback callback function used on some drivers to call either the tuner or the demodulator.

id Frontend ID

exit Used to inform the DVB core that the frontend thread should exit (usually, means that the hardware got disconnected.

int **dvb_register_frontend**(struct *dvb_adapter* * *dvb*, struct *dvb_frontend* * *fe*)
Registers a DVB frontend at the adapter

Parameters

struct dvb_adapter * dvb pointer to the dvb adapter

struct dvb_frontend * fe pointer to the frontend struct

Description

Allocate and initialize the private data needed by the frontend core to manage the frontend and calls *dvb_register_device()* to register a new frontend. It also cleans the property cache that stores the frontend parameters and selects the first available delivery system.

int **dvb_unregister_frontend**(struct *dvb_frontend* * *fe*)
Unregisters a DVB frontend

Parameters

struct dvb_frontend * fe pointer to the frontend struct

Description

Stops the frontend kthread, calls *dvb_unregister_device()* and frees the private frontend data allocated by *dvb_register_frontend()*.

NOTE

This function doesn't frees the memory allocated by the demod, by the SEC driver and by the tuner. In order to free it, an explicit call to *dvb_frontend_detach()* is needed, after calling this function.

void **dvb_frontend_detach**(struct *dvb_frontend* * *fe*)
Detaches and frees frontend specific data

Parameters

struct dvb_frontend * fe pointer to the frontend struct

Description

This function should be called after *dvb_unregister_frontend()*. It calls the SEC, tuner and demod release functions: *dvb_frontend_ops.release_sec*, *dvb_frontend_ops.tuner_ops*.release, *dvb_frontend_ops.analog_ops*.release and *dvb_frontend_ops.release*.

If the driver is compiled with CONFIG_MEDIA_ATTACH, it also decreases the module reference count, needed to allow userspace to remove the previously used DVB frontend modules.

int **dvb_frontend_suspend**(struct *dvb_frontend* * *fe*)
Suspends a Digital TV frontend

Parameters

struct dvb_frontend * fe pointer to the frontend struct

Description

This function prepares a Digital TV frontend to suspend.

In order to prepare the tuner to suspend, if *dvb_frontend_ops.tuner_ops*.suspend() is available, it calls it. Otherwise, it will call *dvb_frontend_ops.tuner_ops*.sleep(), if available.

It will also call *dvb_frontend_ops.sleep*() to put the demod to suspend.

The drivers should also call dvb_frontend_suspend() as part of their handler for the device_driver.suspend().

int **dvb_frontend_resume**(struct *dvb_frontend* * *fe*)
Resumes a Digital TV frontend

Parameters

struct dvb_frontend * fe pointer to the frontend struct

Description

This function resumes the usual operation of the tuner after resume.

In order to resume the frontend, it calls the demod *dvb_frontend_ops.init*().

If *dvb_frontend_ops.tuner_ops*.resume() is available, It, it calls it. Otherwise,t will call *dvb_frontend_ops.tuner_ops*.init(), if available.

Once tuner and demods are resumed, it will enforce that the SEC voltage and tone are restored to their previous values and wake up the frontend's kthread in order to retune the frontend.

The drivers should also call *dvb_frontend_resume()* as part of their handler for the device_driver.resume().

void **dvb_frontend_reinitialise**(struct *dvb_frontend* * *fe*)
forces a reinitialisation at the frontend

Parameters

struct dvb_frontend * fe pointer to the frontend struct

Description

Calls *dvb_frontend_ops.init*() and *dvb_frontend_ops.tuner_ops*.init(), and resets SEC tone and voltage (for Satellite systems).

NOTE

Currently, this function is used only by one driver (budget-av). It seems to be due to address some special issue with that specific frontend.

void **dvb_frontend_sleep_until**(ktime_t * *waketime*, u32 *add_usec*)
Sleep for the amount of time given by add_usec parameter

Parameters

ktime_t * waketime pointer to a struct ktime_t

u32 add_usec time to sleep, in microseconds

Description

This function is used to measure the time required for the FE_DISHNETWORK_SEND_LEGACY_CMD ioctl to work. It needs to be as precise as possible, as it affects the detection of the dish tone command at the satellite subsystem.

Its used internally by the DVB frontend core, in order to emulate FE_DISHNETWORK_SEND_LEGACY_CMD using the *dvb_frontend_ops.set_voltage*() callback.

NOTE

it should not be used at the drivers, as the emulation for the legacy callback is provided by the Kernel. The only situation where this should be at the drivers is when there are some bugs at the hardware that would prevent the core emulation to work. On such cases, the driver would be writing a *dvb_frontend_ops.dishnetwork_send_legacy_command*() and calling this function directly.

2.6 Digital TV Demux kABI

2.6.1 Digital TV Demux

The Kernel Digital TV Demux kABI defines a driver-internal interface for registering low-level, hardware specific driver to a hardware independent demux layer. It is only of interest for Digital TV device driver writers. The header file for this kABI is named demux.h and located in drivers/media/dvb-core.

The demux kABI should be implemented for each demux in the system. It is used to select the TS source of a demux and to manage the demux resources. When the demux client allocates a resource via the demux kABI, it receives a pointer to the kABI of that resource.

Each demux receives its TS input from a DVB front-end or from memory, as set via this demux kABI. In a system with more than one front-end, the kABI can be used to select one of the DVB front-ends as a TS source for a demux, unless this is fixed in the HW platform.

The demux kABI only controls front-ends regarding to their connections with demuxes; the kABI used to set the other front-end parameters, such as tuning, are devined via the Digital TV Frontend kABI.

The functions that implement the abstract interface demux should be defined static or module private and registered to the Demux core for external access. It is not necessary to implement every function in the struct &dmx_demux. For example, a demux interface might support Section filtering, but not PES filtering. The kABI client is expected to check the value of any function pointer before calling the function: the value of `NULL` means that the function is not available.

Whenever the functions of the demux API modify shared data, the possibilities of lost update and race condition problems should be addressed, e.g. by protecting parts of code with mutexes.

Note that functions called from a bottom half context must not sleep. Even a simple memory allocation without using `GFP_ATOMIC` can result in a kernel thread being put to sleep if swapping is needed. For example, the Linux Kernel calls the functions of a network device interface from a bottom half context. Thus, if a demux kABI function is called from network device code, the function must not sleep.

2.7 Demux Callback API

2.7.1 Demux Callback

This kernel-space API comprises the callback functions that deliver filtered data to the demux client. Unlike the other DVB kABIs, these functions are provided by the client and called from the demux code.

The function pointers of this abstract interface are not packed into a structure as in the other demux APIs, because the callback functions are registered and used independent of each other. As an example, it is possible for the API client to provide several callback functions for receiving TS packets and no callbacks for PES packets or sections.

The functions that implement the callback API need not be re-entrant: when a demux driver calls one of these functions, the driver is not allowed to call the function again before the original call returns. If a callback is triggered by a hardware interrupt, it is recommended to use the Linux bottom half mechanism or start a tasklet instead of making the callback function call directly from a hardware interrupt.

This mechanism is implemented by *dmx_ts_cb()* and *dmx_section_cb()* callbacks.

enum **ts_filter_type**
filter type bitmap for dmx_ts_feed.set()

Constants

TS_PACKET Send TS packets (188 bytes) to callback (default).

TS_PAYLOAD_ONLY In case TS_PACKET is set, only send the TS payload (<=184 bytes per packet) to callback

TS_DECODER Send stream to built-in decoder (if present).

TS_DEMUX In case TS_PACKET is set, send the TS to the demux device, not to the dvr device

struct **dmx_ts_feed**
Structure that contains a TS feed filter

Definition

```
struct dmx_ts_feed {
  int is_filtering;
  struct dmx_demux * parent;
  void * priv;
  int (* set) (struct dmx_ts_feed *feed,u16 pid,int type,enum dmx_ts_pes pes_type,ktime_t␣
 ↪timeout);
  int (* start_filtering) (struct dmx_ts_feed *feed);
  int (* stop_filtering) (struct dmx_ts_feed *feed);
};
```

Members

is_filtering Set to non-zero when filtering in progress

parent pointer to struct dmx_demux

priv pointer to private data of the API client

set sets the TS filter

start_filtering starts TS filtering

stop_filtering stops TS filtering

Description

A TS feed is typically mapped to a hardware PID filter on the demux chip. Using this API, the client can set the filtering properties to start/stop filtering TS packets on a particular TS feed.

struct **dmx_section_filter**
Structure that describes a section filter

Definition

```
struct dmx_section_filter {
  u8 filter_value[DMX_MAX_FILTER_SIZE];
  u8 filter_mask[DMX_MAX_FILTER_SIZE];
  u8 filter_mode[DMX_MAX_FILTER_SIZE];
  struct dmx_section_feed * parent;
  void * priv;
};
```

Members

filter_value[DMX_MAX_FILTER_SIZE] Contains up to 16 bytes (128 bits) of the TS section header that will be matched by the section filter

filter_mask[DMX_MAX_FILTER_SIZE] Contains a 16 bytes (128 bits) filter mask with the bits specified by **filter_value** that will be used on the filter match logic.

filter_mode[DMX_MAX_FILTER_SIZE] Contains a 16 bytes (128 bits) filter mode.

parent Pointer to struct dmx_section_feed.

priv Pointer to private data of the API client.

Description

The **filter_mask** controls which bits of **filter_value** are compared with the section headers/payload. On a binary value of 1 in filter_mask, the corresponding bits are compared. The filter only accepts sections that are equal to filter_value in all the tested bit positions.

struct **dmx_section_feed**
Structure that contains a section feed filter

Definition

```
struct dmx_section_feed {
  int is_filtering;
  struct dmx_demux * parent;
  void * priv;
  int check_crc;
  int (* set) (struct dmx_section_feed *feed,u16 pid,int check_crc);
  int (* allocate_filter) (struct dmx_section_feed *feed,struct dmx_section_filter **filter);
  int (* release_filter) (struct dmx_section_feed *feed,struct dmx_section_filter *filter);
  int (* start_filtering) (struct dmx_section_feed *feed);
  int (* stop_filtering) (struct dmx_section_feed *feed);
};
```

Members

is_filtering Set to non-zero when filtering in progress

parent pointer to struct dmx_demux

priv pointer to private data of the API client

check_crc If non-zero, check the CRC values of filtered sections.

set sets the section filter

allocate_filter This function is used to allocate a section filter on the demux. It should only be called when no filtering is in progress on this section feed. If a filter cannot be allocated, the function fails with -ENOSPC.

release_filter This function releases all the resources of a previously allocated section filter. The function should not be called while filtering is in progress on this section feed. After calling this function, the caller should not try to dereference the filter pointer.

start_filtering starts section filtering

stop_filtering stops section filtering

Description

A TS feed is typically mapped to a hardware PID filter on the demux chip. Using this API, the client can set the filtering properties to start/stop filtering TS packets on a particular TS feed.

dmx_ts_cb
Typedef: DVB demux TS filter callback function prototype

Syntax

```
int dmx_ts_cb (const u8 * buffer1,size_t buffer1_length,const u8 *
buffer2,size_t buffer2_length,struct dmx_ts_feed * source);
```

Parameters

const u8 * buffer1 Pointer to the start of the filtered TS packets.

size_t buffer1_length Length of the TS data in buffer1.

const u8 * buffer2 Pointer to the tail of the filtered TS packets, or NULL.

size_t buffer2_length Length of the TS data in buffer2.

struct dmx_ts_feed * source Indicates which TS feed is the source of the callback.

Description

This function callback prototype, provided by the client of the demux API, is called from the demux code. The function is only called when filtering on a TS feed has been enabled using the start_filtering() function at the *dmx_demux*. Any TS packets that match the filter settings are copied to a circular buffer. The filtered TS packets are delivered to the client using this callback function. It is expected that the **buffer1** and **buffer2** callback parameters point to addresses within the circular buffer, but other implementations are also possible. Note that the called party should not try to free the memory the **buffer1** and **buffer2** parameters point to.

When this function is called, the **buffer1** parameter typically points to the start of the first undelivered TS packet within a circular buffer. The **buffer2** buffer parameter is normally NULL, except when the received TS packets have crossed the last address of the circular buffer and ”wrapped” to the beginning of the buffer. In the latter case the **buffer1** parameter would contain an address within the circular buffer, while the **buffer2** parameter would contain the first address of the circular buffer. The number of bytes delivered with this function (i.e. **buffer1_length** + **buffer2_length**) is usually equal to the value of callback_length parameter given in the `set()` function, with one exception: if a timeout occurs before receiving callback_length bytes of TS data, any undelivered packets are immediately delivered to the client by calling this function. The timeout duration is controlled by the `set()` function in the TS Feed API.

If a TS packet is received with errors that could not be fixed by the TS-level forward error correction (FEC), the Transport_error_indicator flag of the TS packet header should be set. The TS packet should not be discarded, as the error can possibly be corrected by a higher layer protocol. If the called party is slow in processing the callback, it is possible that the circular buffer eventually fills up. If this happens, the demux driver should discard any TS packets received while the buffer is full and return -EOVERFLOW.

The type of data returned to the callback can be selected by the *dmx_ts_feed*.**set** function. The type parameter decides if the raw TS packet (TS_PACKET) or just the payload (TS_PACKET|TS_PAYLOAD_ONLY) should be returned. If additionally the TS_DECODER bit is set the stream will also be sent to the hardware MPEG decoder.

Return

- 0, on success;
- -EOVERFLOW, on buffer overflow.

dmx_section_cb
Typedef: DVB demux TS filter callback function prototype

Syntax

```
int dmx_section_cb (const u8 * buffer1,size_t buffer1_len,const u8 *
buffer2,size_t buffer2_len,struct dmx_section_filter * source);
```

Parameters

const u8 * buffer1 Pointer to the start of the filtered section, e.g. within the circular buffer of the demux driver.

size_t buffer1_len Length of the filtered section data in **buffer1**, including headers and CRC.

const u8 * buffer2 Pointer to the tail of the filtered section data, or NULL. Useful to handle the wrapping of a circular buffer.

size_t buffer2_len Length of the filtered section data in **buffer2**, including headers and CRC.

struct dmx_section_filter * source Indicates which section feed is the source of the callback.

Description

This function callback prototype, provided by the client of the demux API, is called from the demux code. The function is only called when filtering of sections has been enabled using the function *dmx_ts_feed*.**start_filtering**. When the demux driver has received a complete section that matches

at least one section filter, the client is notified via this callback function. Normally this function is called for each received section; however, it is also possible to deliver multiple sections with one callback, for example when the system load is high. If an error occurs while receiving a section, this function should be called with the corresponding error type set in the success field, whether or not there is data to deliver. The Section Feed implementation should maintain a circular buffer for received sections. However, this is not necessary if the Section Feed API is implemented as a client of the TS Feed API, because the TS Feed implementation then buffers the received data. The size of the circular buffer can be configured using the *dmx_ts_feed*.**set** function in the Section Feed API. If there is no room in the circular buffer when a new section is received, the section must be discarded. If this happens, the value of the success parameter should be DMX_OVERRUN_ERROR on the next callback.

enum **dmx_frontend_source**
Used to identify the type of frontend

Constants

DMX_MEMORY_FE The source of the demux is memory. It means that the MPEG-TS to be filtered comes from userspace, via *write()* syscall.

DMX_FRONTEND_0 The source of the demux is a frontend connected to the demux.

struct **dmx_frontend**
Structure that lists the frontends associated with a demux

Definition

```
struct dmx_frontend {
  struct list_head connectivity_list;
  enum dmx_frontend_source source;
};
```

Members

connectivity_list List of front-ends that can be connected to a particular demux;

source Type of the frontend.

Description

FIXME: this structure should likely be replaced soon by some media-controller based logic.

enum **dmx_demux_caps**
MPEG-2 TS Demux capabilities bitmap

Constants

DMX_TS_FILTERING set if TS filtering is supported;

DMX_SECTION_FILTERING set if section filtering is supported;

DMX_MEMORY_BASED_FILTERING set if *write()* available.

Description

Those flags are OR'ed in the *dmx_demux.capabilities* field

DMX_FE_ENTRY(*list*)
Casts elements in the list of registered front-ends from the generic type struct list_head to the type * struct dmx_frontend

Parameters

list list of struct dmx_frontend

struct **dmx_demux**
Structure that contains the demux capabilities and callbacks.

Definition

```
struct dmx_demux {
  enum dmx_demux_caps capabilities;
  struct dmx_frontend * frontend;
  void * priv;
  int (* open) (struct dmx_demux *demux);
  int (* close) (struct dmx_demux *demux);
  int (* write) (struct dmx_demux *demux, const char __user *buf,size_t count);
  int (* allocate_ts_feed) (struct dmx_demux *demux,struct dmx_ts_feed **feed,dmx_ts_cb␣
 ↪callback);
  int (* release_ts_feed) (struct dmx_demux *demux,struct dmx_ts_feed *feed);
  int (* allocate_section_feed) (struct dmx_demux *demux,struct dmx_section_feed **feed,dmx_
 ↪section_cb callback);
  int (* release_section_feed) (struct dmx_demux *demux,struct dmx_section_feed *feed);
  int (* add_frontend) (struct dmx_demux *demux,struct dmx_frontend *frontend);
  int (* remove_frontend) (struct dmx_demux *demux,struct dmx_frontend *frontend);
  struct list_head *(* get_frontends) (struct dmx_demux *demux);
  int (* connect_frontend) (struct dmx_demux *demux,struct dmx_frontend *frontend);
  int (* disconnect_frontend) (struct dmx_demux *demux);
  int (* get_pes_pids) (struct dmx_demux *demux, u16 *pids);
};
```

Members

capabilities Bitfield of capability flags.

frontend Front-end connected to the demux

priv Pointer to private data of the API client

open This function reserves the demux for use by the caller and, if necessary, initializes the demux. When the demux is no longer needed, the function **close** should be called. It should be possible for multiple clients to access the demux at the same time. Thus, the function implementation should increment the demux usage count when **open** is called and decrement it when **close** is called. The **demux** function parameter contains a pointer to the demux API and instance data. It returns: 0 on success; -EUSERS, if maximum usage count was reached; -EINVAL, on bad parameter.

close This function reserves the demux for use by the caller and, if necessary, initializes the demux. When the demux is no longer needed, the function **close** should be called. It should be possible for multiple clients to access the demux at the same time. Thus, the function implementation should increment the demux usage count when **open** is called and decrement it when **close** is called. The **demux** function parameter contains a pointer to the demux API and instance data. It returns: 0 on success; -ENODEV, if demux was not in use (e. g. no users); -EINVAL, on bad parameter.

write This function provides the demux driver with a memory buffer containing TS packets. Instead of receiving TS packets from the DVB front-end, the demux driver software will read packets from memory. Any clients of this demux with active TS, PES or Section filters will receive filtered data via the Demux callback API (see 0). The function returns when all the data in the buffer has been consumed by the demux. Demux hardware typically cannot read TS from memory. If this is the case, memory-based filtering has to be implemented entirely in software. The **demux** function parameter contains a pointer to the demux API and instance data. The **buf** function parameter contains a pointer to the TS data in kernel-space memory. The **count** function parameter contains the length of the TS data. It returns: 0 on success; -ERESTARTSYS, if mutex lock was interrupted; -EINTR, if a signal handling is pending; -ENODEV, if demux was removed; -EINVAL, on bad parameter.

allocate_ts_feed Allocates a new TS feed, which is used to filter the TS packets carrying a certain PID. The TS feed normally corresponds to a hardware PID filter on the demux chip. The **demux** function parameter contains a pointer to the demux API and instance data. The **feed** function parameter contains a pointer to the TS feed API and instance data. The **callback** function parameter contains a pointer to the callback function for passing received TS packet. It returns: 0 on success; -ERESTARTSYS, if mutex lock was interrupted; -EBUSY, if no more TS feeds is available; -EINVAL, on bad parameter.

release_ts_feed Releases the resources allocated with **allocate_ts_feed**. Any filtering in progress on

the TS feed should be stopped before calling this function. The **demux** function parameter contains a pointer to the demux API and instance data. The **feed** function parameter contains a pointer to the TS feed API and instance data. It returns: 0 on success; -EINVAL on bad parameter.

allocate_section_feed Allocates a new section feed, i.e. a demux resource for filtering and receiving sections. On platforms with hardware support for section filtering, a section feed is directly mapped to the demux HW. On other platforms, TS packets are first PID filtered in hardware and a hardware section filter then emulated in software. The caller obtains an API pointer of type dmx_section_feed_t as an out parameter. Using this API the caller can set filtering parameters and start receiving sections. The **demux** function parameter contains a pointer to the demux API and instance data. The **feed** function parameter contains a pointer to the TS feed API and instance data. The **callback** function parameter contains a pointer to the callback function for passing received TS packet. It returns: 0 on success; -EBUSY, if no more TS feeds is available; -EINVAL, on bad parameter.

release_section_feed Releases the resources allocated with **allocate_section_feed**, including allocated filters. Any filtering in progress on the section feed should be stopped before calling this function. The **demux** function parameter contains a pointer to the demux API and instance data. The **feed** function parameter contains a pointer to the TS feed API and instance data. It returns: 0 on success; -EINVAL, on bad parameter.

add_frontend Registers a connectivity between a demux and a front-end, i.e., indicates that the demux can be connected via a call to **connect_frontend** to use the given front-end as a TS source. The client of this function has to allocate dynamic or static memory for the frontend structure and initialize its fields before calling this function. This function is normally called during the driver initialization. The caller must not free the memory of the frontend struct before successfully calling **remove_frontend**. The **demux** function parameter contains a pointer to the demux API and instance data. The **frontend** function parameter contains a pointer to the front-end instance data. It returns: 0 on success; -EINVAL, on bad parameter.

remove_frontend Indicates that the given front-end, registered by a call to **add_frontend**, can no longer be connected as a TS source by this demux. The function should be called when a front-end driver or a demux driver is removed from the system. If the front-end is in use, the function fails with the return value of -EBUSY. After successfully calling this function, the caller can free the memory of the frontend struct if it was dynamically allocated before the **add_frontend** operation. The **demux** function parameter contains a pointer to the demux API and instance data. The **frontend** function parameter contains a pointer to the front-end instance data. It returns: 0 on success; -ENODEV, if the front-end was not found, -EINVAL, on bad parameter.

get_frontends Provides the APIs of the front-ends that have been registered for this demux. Any of the front-ends obtained with this call can be used as a parameter for **connect_frontend**. The include file demux.h contains the macro *DMX_FE_ENTRY()* for converting an element of the generic type struct list_head * to the type struct *dmx_frontend . The caller must not free the memory of any of the elements obtained via this function call. The **demux** function parameter contains a pointer to the demux API and instance data. It returns a struct list_head pointer to the list of front-end interfaces, or NULL in the case of an empty list.

connect_frontend Connects the TS output of the front-end to the input of the demux. A demux can only be connected to a front-end registered to the demux with the function **add_frontend**. It may or may not be possible to connect multiple demuxes to the same front-end, depending on the capabilities of the HW platform. When not used, the front-end should be released by calling **disconnect_frontend**. The **demux** function parameter contains a pointer to the demux API and instance data. The **frontend** function parameter contains a pointer to the front-end instance data. It returns: 0 on success; -EINVAL, on bad parameter.

disconnect_frontend Disconnects the demux and a front-end previously connected by a **connect_frontend** call. The **demux** function parameter contains a pointer to the demux API and instance data. It returns: 0 on success; -EINVAL on bad parameter.

get_pes_pids Get the PIDs for DMX_PES_AUDIO0, DMX_PES_VIDEO0, DMX_PES_TELETEXT0, DMX_PES_SUBTITLE0 and DMX_PES_PCR0. The **demux** function parameter contains a pointer to the demux API and instance data. The **pids** function parameter contains an array with five u16 elements where the PIDs will be stored. It returns: 0 on success; -EINVAL on bad parameter.

2.8 Digital TV Conditional Access kABI

struct **dvb_ca_en50221**
Structure describing a CA interface

Definition

```
struct dvb_ca_en50221 {
  struct module * owner;
  int (* read_attribute_mem) (struct dvb_ca_en50221 *ca,int slot, int address);
  int (* write_attribute_mem) (struct dvb_ca_en50221 *ca,int slot, int address, u8 value);
  int (* read_cam_control) (struct dvb_ca_en50221 *ca,int slot, u8 address);
  int (* write_cam_control) (struct dvb_ca_en50221 *ca,int slot, u8 address, u8 value);
  int (* slot_reset) (struct dvb_ca_en50221 *ca, int slot);
  int (* slot_shutdown) (struct dvb_ca_en50221 *ca, int slot);
  int (* slot_ts_enable) (struct dvb_ca_en50221 *ca, int slot);
  int (* poll_slot_status) (struct dvb_ca_en50221 *ca, int slot, int open);
  void * data;
  void * private;
};
```

Members

owner the module owning this structure

read_attribute_mem function for reading attribute memory on the CAM

write_attribute_mem function for writing attribute memory on the CAM

read_cam_control function for reading the control interface on the CAM

write_cam_control function for reading the control interface on the CAM

slot_reset function to reset the CAM slot

slot_shutdown function to shutdown a CAM slot

slot_ts_enable function to enable the Transport Stream on a CAM slot

poll_slot_status function to poll slot status. Only necessary if DVB_CA_FLAG_EN50221_IRQ_CAMCHANGE is not set.

data private data, used by caller.

private Opaque data used by the dvb_ca core. Do not modify!

NOTE

the read_*, write_* and poll_slot_status functions will be called for different slots concurrently and need to use locks where and if appropriate. There will be no concurrent access to one slot.

void **dvb_ca_en50221_camchange_irq**(struct *dvb_ca_en50221 * pubca*, int *slot*, int *change_type*)
A CAMCHANGE IRQ has occurred.

Parameters

struct dvb_ca_en50221 * pubca CA instance.

int slot Slot concerned.

int change_type One of the DVB_CA_CAMCHANGE_* values

void **dvb_ca_en50221_camready_irq**(struct *dvb_ca_en50221 * pubca*, int *slot*)
A CAMREADY IRQ has occurred.

Parameters

struct dvb_ca_en50221 * pubca CA instance.

int slot Slot concerned.

void **dvb_ca_en50221_frda_irq**(struct *dvb_ca_en50221 * ca*, int *slot*)
An FR or a DA IRQ has occurred.

Parameters

struct dvb_ca_en50221 * ca CA instance.

int slot Slot concerned.

int **dvb_ca_en50221_init**(struct *dvb_adapter * dvb_adapter*, struct *dvb_ca_en50221 * ca*, int *flags*, int *slot_count*)
Initialise a new DVB CA device.

Parameters

struct dvb_adapter * dvb_adapter DVB adapter to attach the new CA device to.

struct dvb_ca_en50221 * ca The dvb_ca instance.

int flags Flags describing the CA device (DVB_CA_EN50221_FLAG_*).

int slot_count Number of slots supported.

Description

return 0 on success, nonzero on failure

void **dvb_ca_en50221_release**(struct *dvb_ca_en50221 * ca*)
Release a DVB CA device.

Parameters

struct dvb_ca_en50221 * ca The associated dvb_ca instance.

2.9 Remote Controller devices

2.9.1 Remote Controller core

enum **rc_driver_type**
type of the RC output

Constants

RC_DRIVER_SCANCODE Driver or hardware generates a scancode

RC_DRIVER_IR_RAW Driver or hardware generates pulse/space sequences. It needs a Infra-Red pulse/space decoder

RC_DRIVER_IR_RAW_TX Device transmitter only, driver requires pulse/space data sequence.

struct **rc_scancode_filter**
Filter scan codes.

Definition

```
struct rc_scancode_filter {
  u32 data;
  u32 mask;
};
```

Members

data Scancode data to match.

mask Mask of bits of scancode to compare.

enum **rc_filter_type**
Filter type constants.

Constants

RC_FILTER_NORMAL Filter for normal operation.

RC_FILTER_WAKEUP Filter for waking from suspend.

RC_FILTER_MAX Number of filter types.

struct **rc_dev**
represents a remote control device

Definition

```
struct rc_dev {
  struct device dev;
  atomic_t initialized;
  bool managed_alloc;
  const struct attribute_group * sysfs_groups[5];
  const char * input_name;
  const char * input_phys;
  struct input_id input_id;
  char * driver_name;
  const char * map_name;
  struct rc_map rc_map;
  struct mutex lock;
  unsigned int minor;
  struct ir_raw_event_ctrl * raw;
  struct input_dev * input_dev;
  enum rc_driver_type driver_type;
  bool idle;
  bool encode_wakeup;
  u64 allowed_protocols;
  u64 enabled_protocols;
  u64 allowed_wakeup_protocols;
  enum rc_type wakeup_protocol;
  struct rc_scancode_filter scancode_filter;
  struct rc_scancode_filter scancode_wakeup_filter;
  u32 scancode_mask;
  u32 users;
  void * priv;
  spinlock_t keylock;
  bool keypressed;
  unsigned long keyup_jiffies;
  struct timer_list timer_keyup;
  u32 last_keycode;
  enum rc_type last_protocol;
  u32 last_scancode;
  u8 last_toggle;
  u32 timeout;
  u32 min_timeout;
  u32 max_timeout;
  u32 rx_resolution;
  u32 tx_resolution;
  int (* change_protocol) (struct rc_dev *dev, u64 *rc_type);
  int (* open) (struct rc_dev *dev);
  void (* close) (struct rc_dev *dev);
  int (* s_tx_mask) (struct rc_dev *dev, u32 mask);
  int (* s_tx_carrier) (struct rc_dev *dev, u32 carrier);
  int (* s_tx_duty_cycle) (struct rc_dev *dev, u32 duty_cycle);
  int (* s_rx_carrier_range) (struct rc_dev *dev, u32 min, u32 max);
  int (* tx_ir) (struct rc_dev *dev, unsigned *txbuf, unsigned n);
  void (* s_idle) (struct rc_dev *dev, bool enable);
  int (* s_learning_mode) (struct rc_dev *dev, int enable);
  int (* s_carrier_report) (struct rc_dev *dev, int enable);
  int (* s_filter) (struct rc_dev *dev,struct rc_scancode_filter *filter);
```

```
  int (* s_wakeup_filter) (struct rc_dev *dev,struct rc_scancode_filter *filter);
  int (* s_timeout) (struct rc_dev *dev,unsigned int timeout);
};
```

Members

dev driver model's view of this device

initialized 1 if the device init has completed, 0 otherwise

managed_alloc devm_rc_allocate_device was used to create rc_dev

sysfs_groups[5] sysfs attribute groups

input_name name of the input child device

input_phys physical path to the input child device

input_id id of the input child device (struct input_id)

driver_name name of the hardware driver which registered this device

map_name name of the default keymap

rc_map current scan/key table

lock used to ensure we've filled in all protocol details before anyone can call show_protocols or store_protocols

minor unique minor remote control device number

raw additional data for raw pulse/space devices

input_dev the input child device used to communicate events to userspace

driver_type specifies if protocol decoding is done in hardware or software

idle used to keep track of RX state

encode_wakeup wakeup filtering uses IR encode API, therefore the allowed wakeup protocols is the set of all raw encoders

allowed_protocols bitmask with the supported RC_BIT_* protocols

enabled_protocols bitmask with the enabled RC_BIT_* protocols

allowed_wakeup_protocols bitmask with the supported RC_BIT_* wakeup protocols

wakeup_protocol the enabled RC_TYPE_* wakeup protocol or RC_TYPE_UNKNOWN if disabled.

scancode_filter scancode filter

scancode_wakeup_filter scancode wakeup filters

scancode_mask some hardware decoders are not capable of providing the full scancode to the application. As this is a hardware limit, we can't do anything with it. Yet, as the same keycode table can be used with other devices, a mask is provided to allow its usage. Drivers should generally leave this field in blank

users number of current users of the device

priv driver-specific data

keylock protects the remaining members of the struct

keypressed whether a key is currently pressed

keyup_jiffies time (in jiffies) when the current keypress should be released

timer_keyup timer for releasing a keypress

last_keycode keycode of last keypress

last_protocol protocol of last keypress

last_scancode scancode of last keypress

last_toggle toggle value of last command

timeout optional time after which device stops sending data

min_timeout minimum timeout supported by device

max_timeout maximum timeout supported by device

rx_resolution resolution (in ns) of input sampler

tx_resolution resolution (in ns) of output sampler

change_protocol allow changing the protocol used on hardware decoders

open callback to allow drivers to enable polling/irq when IR input device is opened.

close callback to allow drivers to disable polling/irq when IR input device is opened.

s_tx_mask set transmitter mask (for devices with multiple tx outputs)

s_tx_carrier set transmit carrier frequency

s_tx_duty_cycle set transmit duty cycle (0% - 100%)

s_rx_carrier_range inform driver about carrier it is expected to handle

tx_ir transmit IR

s_idle enable/disable hardware idle mode, upon which, device doesn't interrupt host until it sees IR pulses

s_learning_mode enable wide band receiver used for learning

s_carrier_report enable carrier reports

s_filter set the scancode filter

s_wakeup_filter set the wakeup scancode filter. If the mask is zero then wakeup should be disabled. wakeup_protocol will be set to a valid protocol if mask is nonzero.

s_timeout set hardware timeout in ns

struct *rc_dev* * **rc_allocate_device**(enum *rc_driver_type*)
Allocates a RC device

Parameters

enum rc_driver_type specifies the type of the RC output to be allocated returns a pointer to struct rc_dev.

struct *rc_dev* * **devm_rc_allocate_device**(struct device * *dev*, enum *rc_driver_type*)
Managed RC device allocation

Parameters

struct device * dev pointer to struct device

enum rc_driver_type specifies the type of the RC output to be allocated returns a pointer to struct rc_dev.

void **rc_free_device**(struct *rc_dev* * *dev*)
Frees a RC device

Parameters

struct rc_dev * dev pointer to struct rc_dev.

int **rc_register_device**(struct *rc_dev* * *dev*)
Registers a RC device

Parameters

struct rc_dev * dev pointer to struct rc_dev.

int **devm_rc_register_device**(struct device * *parent*, struct *rc_dev* * *dev*)
Manageded registering of a RC device

Parameters

struct device * parent pointer to struct device.

struct rc_dev * dev pointer to struct rc_dev.

void **rc_unregister_device**(struct *rc_dev* * *dev*)
Unregisters a RC device

Parameters

struct rc_dev * dev pointer to struct rc_dev.

int **rc_open**(struct *rc_dev* * *rdev*)
Opens a RC device

Parameters

struct rc_dev * rdev pointer to struct rc_dev.

void **rc_close**(struct *rc_dev* * *rdev*)
Closes a RC device

Parameters

struct rc_dev * rdev pointer to struct rc_dev.

enum **rc_type**
type of the Remote Controller protocol

Constants

RC_TYPE_UNKNOWN Protocol not known

RC_TYPE_OTHER Protocol known but proprietary

RC_TYPE_RC5 Philips RC5 protocol

RC_TYPE_RC5X_20 Philips RC5x 20 bit protocol

RC_TYPE_RC5_SZ StreamZap variant of RC5

RC_TYPE_JVC JVC protocol

RC_TYPE_SONY12 Sony 12 bit protocol

RC_TYPE_SONY15 Sony 15 bit protocol

RC_TYPE_SONY20 Sony 20 bit protocol

RC_TYPE_NEC NEC protocol

RC_TYPE_NECX Extended NEC protocol

RC_TYPE_NEC32 NEC 32 bit protocol

RC_TYPE_SANYO Sanyo protocol

RC_TYPE_MCE_KBD RC6-ish MCE keyboard/mouse

RC_TYPE_RC6_0 Philips RC6-0-16 protocol

RC_TYPE_RC6_6A_20 Philips RC6-6A-20 protocol

RC_TYPE_RC6_6A_24 Philips RC6-6A-24 protocol

RC_TYPE_RC6_6A_32 Philips RC6-6A-32 protocol

RC_TYPE_RC6_MCE MCE (Philips RC6-6A-32 subtype) protocol

RC_TYPE_SHARP Sharp protocol

RC_TYPE_XMP XMP protocol

RC_TYPE_CEC CEC protocol

struct **rc_map_table**
represents a scancode/keycode pair

Definition

```
struct rc_map_table {
  u32 scancode;
  u32 keycode;
};
```

Members

scancode scan code (u32)

keycode Linux input keycode

struct **rc_map**
represents a keycode map table

Definition

```
struct rc_map {
  struct rc_map_table * scan;
  unsigned int size;
  unsigned int len;
  unsigned int alloc;
  enum rc_type rc_type;
  const char * name;
  spinlock_t lock;
};
```

Members

scan pointer to struct *rc_map_table*

size Max number of entries

len Number of entries that are in use

alloc size of *scan, in bytes

rc_type type of the remote controller protocol, as defined at enum *rc_type*

name name of the key map table

lock lock to protect access to this structure

struct **rc_map_list**
list of the registered *rc_map* maps

Definition

```
struct rc_map_list {
  struct list_head list;
  struct rc_map map;
};
```

Members

list pointer to struct `list_head`

map pointer to struct *rc_map*

int **rc_map_register**(struct *rc_map_list* * *map*)
Registers a Remote Controler scancode map

Parameters

struct rc_map_list * map pointer to struct rc_map_list

void **rc_map_unregister**(struct *rc_map_list * map*)
Unregisters a Remote Controler scancode map

Parameters

struct rc_map_list * map pointer to struct rc_map_list

struct *rc_map* * **rc_map_get**(const char * *name*)
gets an RC map from its name

Parameters

const char * name name of the RC scancode map

2.9.2 LIRC

struct **lirc_driver**
Defines the parameters on a LIRC driver

Definition

```
struct lirc_driver {
  char name[40];
  int minor;
  __u32 code_length;
  unsigned int buffer_size;
  int sample_rate;
  __u32 features;
  unsigned int chunk_size;
  void * data;
  int min_timeout;
  int max_timeout;
  int (* add_to_buf) (void *data, struct lirc_buffer *buf);
  struct lirc_buffer * rbuf;
  int (* set_use_inc) (void *data);
  void (* set_use_dec) (void *data);
  struct rc_dev * rdev;
  const struct file_operations * fops;
  struct device * dev;
  struct module * owner;
};
```

Members

name[40] this string will be used for logs

minor indicates minor device (/dev/lirc) number for registered driver if caller fills it with negative value, then the first free minor number will be used (if available).

code_length length of the remote control key code expressed in bits.

buffer_size Number of FIFO buffers with **chunk_size** size. If zero, creates a buffer with BUFLEN size (16 bytes).

sample_rate if zero, the device will wait for an event with a new code to be parsed. Otherwise, specifies the sample rate for polling. Value should be between 0 and HZ. If equal to HZ, it would mean one polling per second.

features lirc compatible hardware features, like LIRC_MODE_RAW, LIRC_CAN_*, as defined at include/media/lirc.h.

chunk_size Size of each FIFO buffer.

data it may point to any driver data and this pointer will be passed to all callback functions.

min_timeout Minimum timeout for record. Valid only if LIRC_CAN_SET_REC_TIMEOUT is defined.

max_timeout Maximum timeout for record. Valid only if LIRC_CAN_SET_REC_TIMEOUT is defined.

add_to_buf add_to_buf will be called after specified period of the time or triggered by the external event, this behavior depends on value of the sample_rate this function will be called in user context. This routine should return 0 if data was added to the buffer and -ENODATA if none was available. This should add some number of bits evenly divisible by code_length to the buffer.

rbuf if not NULL, it will be used as a read buffer, you will have to write to the buffer by other means, like irq's (see also lirc_serial.c).

set_use_inc set_use_inc will be called after device is opened

set_use_dec set_use_dec will be called after device is closed

rdev Pointed to struct rc_dev associated with the LIRC device.

fops file_operations for drivers which don't fit the current driver model. Some ioctl's can be directly handled by lirc_dev if the driver's ioctl function is NULL or if it returns -ENOIOCTLCMD (see also lirc_serial.c).

dev pointer to the struct device associated with the LIRC device.

owner the module owning this struct

2.10 Media Controller devices

2.10.1 Media Controller

The media controller userspace API is documented in *the Media Controller uAPI book* . This document focus on the kernel-side implementation of the media framework.

Abstract media device model

Discovering a device internal topology, and configuring it at runtime, is one of the goals of the media framework. To achieve this, hardware devices are modelled as an oriented graph of building blocks called entities connected through pads.

An entity is a basic media hardware building block. It can correspond to a large variety of logical blocks such as physical hardware devices (CMOS sensor for instance), logical hardware devices (a building block in a System-on-Chip image processing pipeline), DMA channels or physical connectors.

A pad is a connection endpoint through which an entity can interact with other entities. Data (not restricted to video) produced by an entity flows from the entity's output to one or more entity inputs. Pads should not be confused with physical pins at chip boundaries.

A link is a point-to-point oriented connection between two pads, either on the same entity or on different entities. Data flows from a source pad to a sink pad.

Media device

A media device is represented by a struct *media_device* instance, defined in `include/media/media-device.h`. Allocation of the structure is handled by the media device driver, usually by embedding the *media_device* instance in a larger driver-specific structure.

Drivers register media device instances by calling *__media_device_register()* via the macro `media_device_register()` and unregistered by calling *media_device_unregister()*.

Entities

Entities are represented by a struct *media_entity* instance, defined in include/media/media-entity.h. The structure is usually embedded into a higher-level structure, such as *v4l2_subdev* or *video_device* instances, although drivers can allocate entities directly.

Drivers initialize entity pads by calling *media_entity_pads_init()*.

Drivers register entities with a media device by calling *media_device_register_entity()* and unregistred by calling *media_device_unregister_entity()*.

Interfaces

Interfaces are represented by a struct *media_interface* instance, defined in include/media/media-entity.h. Currently, only one type of interface is defined: a device node. Such interfaces are represented by a struct *media_intf_devnode*.

Drivers initialize and create device node interfaces by calling *media_devnode_create()* and remove them by calling: *media_devnode_remove()*.

Pads

Pads are represented by a struct *media_pad* instance, defined in include/media/media-entity.h. Each entity stores its pads in a pads array managed by the entity driver. Drivers usually embed the array in a driver-specific structure.

Pads are identified by their entity and their 0-based index in the pads array.

Both information are stored in the struct *media_pad*, making the struct *media_pad* pointer the canonical way to store and pass link references.

Pads have flags that describe the pad capabilities and state.

MEDIA_PAD_FL_SINK indicates that the pad supports sinking data. MEDIA_PAD_FL_SOURCE indicates that the pad supports sourcing data.

Note:

One and only one of MEDIA_PAD_FL_SINK or MEDIA_PAD_FL_SOURCE must be set for each pad.

Links

Links are represented by a struct *media_link* instance, defined in include/media/media-entity.h. There are two types of links:

1. pad to pad links:

Associate two entities via their PADs. Each entity has a list that points to all links originating at or targeting any of its pads. A given link is thus stored twice, once in the source entity and once in the target entity.

Drivers create pad to pad links by calling: *media_create_pad_link()* and remove with *media_entity_remove_links()*.

2. interface to entity links:

Associate one interface to a Link.

Drivers create interface to entity links by calling: *media_create_intf_link()* and remove with *media_remove_intf_links()*.

Note:

Links can only be created after having both ends already created.

Links have flags that describe the link capabilities and state. The valid values are described at *media_create_pad_link()* and *media_create_intf_link()*.

Graph traversal

The media framework provides APIs to iterate over entities in a graph.

To iterate over all entities belonging to a media device, drivers can use the media_device_for_each_entity macro, defined in include/media/media-device.h.

```
struct media_entity *entity;

media_device_for_each_entity(entity, mdev) {
// entity will point to each entity in turn
...
}
```

Drivers might also need to iterate over all entities in a graph that can be reached only through enabled links starting at a given entity. The media framework provides a depth-first graph traversal API for that purpose.

Note:

*Graphs with cycles (whether directed or undirected) are **NOT** supported by the graph traversal API. To prevent infinite loops, the graph traversal code limits the maximum depth to MEDIA_ENTITY_ENUM_MAX_DEPTH, currently defined as 16.*

Drivers initiate a graph traversal by calling *media_graph_walk_start()*

The graph structure, provided by the caller, is initialized to start graph traversal at the given entity.

Drivers can then retrieve the next entity by calling *media_graph_walk_next()*

When the graph traversal is complete the function will return NULL.

Graph traversal can be interrupted at any moment. No cleanup function call is required and the graph structure can be freed normally.

Helper functions can be used to find a link between two given pads, or a pad connected to another pad through an enabled link *media_entity_find_link()* and *media_entity_remote_pad()*.

Use count and power handling

Due to the wide differences between drivers regarding power management needs, the media controller does not implement power management. However, the struct *media_entity* includes a use_count field that media drivers can use to track the number of users of every entity for power management needs.

The *media_entity*.use_count field is owned by media drivers and must not be touched by entity drivers. Access to the field must be protected by the *media_device*.graph_mutex lock.

Links setup

Link properties can be modified at runtime by calling *media_entity_setup_link()*.

Pipelines and media streams

When starting streaming, drivers must notify all entities in the pipeline to prevent link states from being modified during streaming by calling *media_pipeline_start()*.

The function will mark all entities connected to the given entity through enabled links, either directly or indirectly, as streaming.

The struct *media_pipeline* instance pointed to by the pipe argument will be stored in every entity in the pipeline. Drivers should embed the struct *media_pipeline* in higher-level pipeline structures and can then access the pipeline through the struct *media_entity* pipe field.

Calls to *media_pipeline_start()* can be nested. The pipeline pointer must be identical for all nested calls to the function.

media_pipeline_start() may return an error. In that case, it will clean up any of the changes it did by itself.

When stopping the stream, drivers must notify the entities with *media_pipeline_stop()*.

If multiple calls to *media_pipeline_start()* have been made the same number of *media_pipeline_stop()* calls are required to stop streaming. The *media_entity*.pipe field is reset to NULL on the last nested stop call.

Link configuration will fail with -EBUSY by default if either end of the link is a streaming entity. Links that can be modified while streaming must be marked with the MEDIA_LNK_FL_DYNAMIC flag.

If other operations need to be disallowed on streaming entities (such as changing entities configuration parameters) drivers can explicitly check the media_entity stream_count field to find out if an entity is streaming. This operation must be done with the media_device graph_mutex held.

Link validation

Link validation is performed by *media_pipeline_start()* for any entity which has sink pads in the pipeline. The *media_entity*.link_validate() callback is used for that purpose. In link_validate() callback, entity driver should check that the properties of the source pad of the connected entity and its own sink pad match. It is up to the type of the entity (and in the end, the properties of the hardware) what matching actually means.

Subsystems should facilitate link validation by providing subsystem specific helper functions to provide easy access for commonly needed information, and in the end provide a way to use driver-specific callbacks.

struct **media_entity_notify**
Media Entity Notify

Definition

```
struct media_entity_notify {
  struct list_head list;
  void * notify_data;
  void (* notify) (struct media_entity *entity, void *notify_data);
};
```

Members

list List head

notify_data Input data to invoke the callback

notify Callback function pointer

Description

Drivers may register a callback to take action when new entities get registered with the media device. This handler is intended for creating links between existing entities and should not create entities and register them.

struct **media_device_ops**
Media device operations

Definition

```
struct media_device_ops {
  int (* link_notify) (struct media_link *link, u32 flags,unsigned int notification);
};
```

Members

link_notify Link state change notification callback. This callback is called with the graph_mutex held.

struct **media_device**
Media device

Definition

```
struct media_device {
  struct device * dev;
  struct media_devnode * devnode;
  char model[32];
  char driver_name[32];
  char serial[40];
  char bus_info[32];
  u32 hw_revision;
  u32 driver_version;
  u64 topology_version;
  u32 id;
  struct ida entity_internal_idx;
  int entity_internal_idx_max;
  struct list_head entities;
  struct list_head interfaces;
  struct list_head pads;
  struct list_head links;
  struct list_head entity_notify;
  struct mutex graph_mutex;
  struct media_graph pm_count_walk;
  void * source_priv;
  int (* enable_source) (struct media_entity *entity,struct media_pipeline *pipe);
  void (* disable_source) (struct media_entity *entity);
  const struct media_device_ops * ops;
};
```

Members

dev Parent device

devnode Media device node

model[32] Device model name

driver_name[32] Optional device driver name. If not set, calls to `MEDIA_IOC_DEVICE_INFO` will return `dev->driver->name`. This is needed for USB drivers for example, as otherwise they'll all appear as if the driver name was "usb".

serial[40] Device serial number (optional)

bus_info[32] Unique and stable device location identifier

hw_revision Hardware device revision

driver_version Device driver version

topology_version Monotonic counter for storing the version of the graph topology. Should be incremented each time the topology changes.

id Unique ID used on the last registered graph object

entity_internal_idx Unique internal entity ID used by the graph traversal algorithms

entity_internal_idx_max Allocated internal entity indices

entities List of registered entities

interfaces List of registered interfaces

pads List of registered pads

links List of registered links

entity_notify List of registered entity_notify callbacks

graph_mutex Protects access to struct media_device data

pm_count_walk Graph walk for power state walk. Access serialised using graph_mutex.

source_priv Driver Private data for enable/disable source handlers

enable_source Enable Source Handler function pointer

disable_source Disable Source Handler function pointer

ops Operation handler callbacks

Description

This structure represents an abstract high-level media device. It allows easy access to entities and provides basic media device-level support. The structure can be allocated directly or embedded in a larger structure.

The parent **dev** is a physical device. It must be set before registering the media device.

model is a descriptive model name exported through sysfs. It doesn't have to be unique.

enable_source is a handler to find source entity for the sink entity and activate the link between them if source entity is free. Drivers should call this handler before accessing the source.

disable_source is a handler to find source entity for the sink entity and deactivate the link between them. Drivers should call this handler to release the source.

Use-case: find tuner entity connected to the decoder entity and check if it is available, and activate the the link between them from **enable_source** and deactivate from **disable_source**.

Note:

Bridge driver is expected to implement and set the handler when `media_device` *is registered or when bridge driver finds the media_device during probe. Bridge driver sets source_priv with information necessary to run* ***enable_source*** *and* ***disable_source*** *handlers. Callers should hold graph_mutex to access and call* ***enable_source*** *and* ***disable_source*** *handlers.*

int **media_entity_enum_init**(struct *media_entity_enum* * *ent_enum*, struct *media_device* * *mdev*)
Initialise an entity enumeration

Parameters

struct media_entity_enum * ent_enum Entity enumeration to be initialised

struct media_device * mdev The related media device

Return

zero on success or a negative error code.

void **media_device_init**(struct *media_device* * *mdev*)
Initializes a media device element

Parameters

struct media_device * mdev pointer to struct *media_device*

Description

This function initializes the media device prior to its registration. The media device initialization and registration is split in two functions to avoid race conditions and make the media device available to user-space before the media graph has been completed.

So drivers need to first initialize the media device, register any entity within the media device, create pad to pad links and then finally register the media device by calling *media_device_register()* as a final step.

void **media_device_cleanup**(struct *media_device* * *mdev*)
Cleanups a media device element

Parameters

struct media_device * mdev pointer to struct *media_device*

Description

This function that will destroy the graph_mutex that is initialized in *media_device_init()*.

int **__media_device_register**(struct *media_device* * *mdev*, struct module * *owner*)
Registers a media device element

Parameters

struct media_device * mdev pointer to struct *media_device*

struct module * owner should be filled with THIS_MODULE

Description

Users, should, instead, call the *media_device_register()* macro.

The caller is responsible for initializing the *media_device* structure before registration. The following fields of *media_device* must be set:

- *media_entity.dev* must point to the parent device (usually a pci_dev, usb_interface or platform_device instance).
- *media_entity.model* must be filled with the device model name as a NUL-terminated UTF-8 string. The device/model revision must not be stored in this field.

The following fields are optional:

- *media_entity.serial* is a unique serial number stored as a NUL-terminated ASCII string. The field is big enough to store a GUID in text form. If the hardware doesn't provide a unique serial number this field must be left empty.
- *media_entity.bus_info* represents the location of the device in the system as a NUL-terminated ASCII string. For PCI/PCIe devices *media_entity.bus_info* must be set to "PCI:" (or "PCIe:") followed by the value of pci_name(). For USB devices,the usb_make_path() function must be used. This field is used by applications to distinguish between otherwise identical devices that don't provide a serial number.
- *media_entity.hw_revision* is the hardware device revision in a driver-specific format. When possible the revision should be formatted with the KERNEL_VERSION() macro.
- *media_entity.driver_version* is formatted with the KERNEL_VERSION() macro. The version minor must be incremented when new features are added to the userspace API without breaking binary compatibility. The version major must be incremented when binary compatibility is broken.

Note:

1. *Upon successful registration a character device named media[0-9]+ is created. The device major and minor numbers are dynamic. The model name is exported as a sysfs attribute.*
2. *Unregistering a media device that hasn't been registered is* ***NOT*** *safe.*

Return

returns zero on success or a negative error code.

media_device_register(*mdev*)
Registers a media device element

Parameters

mdev pointer to struct *media_device*

Description

This macro calls *__media_device_register()* passing THIS_MODULE as the *__media_device_register()* second argument (**owner**).

void **media_device_unregister**(struct *media_device* * *mdev*)
Unregisters a media device element

Parameters

struct media_device * mdev pointer to struct *media_device*

Description

It is safe to call this function on an unregistered (but initialised) media device.

int **media_device_register_entity**(struct *media_device* * *mdev*, struct *media_entity* * *entity*)
registers a media entity inside a previously registered media device.

Parameters

struct media_device * mdev pointer to struct *media_device*

struct media_entity * entity pointer to struct *media_entity* to be registered

Description

Entities are identified by a unique positive integer ID. The media controller framework will such ID automatically. IDs are not guaranteed to be contiguous, and the ID number can change on newer Kernel versions. So, neither the driver nor userspace should hardcode ID numbers to refer to the entities, but, instead, use the framework to find the ID, when needed.

The media_entity name, type and flags fields should be initialized before calling *media_device_register_entity()*. Entities embedded in higher-level standard structures can have some of those fields set by the higher-level framework.

If the device has pads, *media_entity_pads_init()* should be called before this function. Otherwise, the *media_entity.pad* and *media_entity.num_pads* should be zeroed before calling this function.

Entities have flags that describe the entity capabilities and state:

MEDIA_ENT_FL_DEFAULT indicates the default entity for a given type. This can be used to report the default audio and video devices or the default camera sensor.

Note:

Drivers should set the entity function before calling this function. Please notice that the values MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN and MEDIA_ENT_F_UNKNOWN should not be used by the drivers.

void **media_device_unregister_entity**(struct *media_entity* * *entity*)
unregisters a media entity.

Parameters

struct media_entity * entity pointer to struct *media_entity* to be unregistered

Description

All links associated with the entity and all PADs are automatically unregistered from the media_device when this function is called.

Unregistering an entity will not change the IDs of the other entities and the previoully used ID will never be reused for a newly registered entities.

When a media device is unregistered, all its entities are unregistered automatically. No manual entities unregistration is then required.

Note:

The media_entity instance itself must be freed explicitly by the driver if required.

int **media_device_register_entity_notify**(struct *media_device* * *mdev*, struct *media_entity_notify* * *nptr*)
Registers a media entity_notify callback

Parameters

struct media_device * mdev The media device

struct media_entity_notify * nptr The media_entity_notify

Description

Note:

When a new entity is registered, all the registered media_entity_notify callbacks are invoked.

void **media_device_unregister_entity_notify**(struct *media_device* * *mdev*, struct *media_entity_notify* * *nptr*)
Unregister a media entity notify callback

Parameters

struct media_device * mdev The media device

struct media_entity_notify * nptr The media_entity_notify

void **media_device_pci_init**(struct *media_device* * *mdev*, struct pci_dev * *pci_dev*, const char * *name*)
create and initialize a struct *media_device* from a PCI device.

Parameters

struct media_device * mdev pointer to struct *media_device*

struct pci_dev * pci_dev pointer to struct pci_dev

const char * name media device name. If NULL, the routine will use the default name for the pci device, given by `pci_name()` macro.

void **__media_device_usb_init**(struct *media_device* * *mdev*, struct usb_device * *udev*, const char * *board_name*, const char * *driver_name*)
create and initialize a struct *media_device* from a PCI device.

Parameters

struct media_device * mdev pointer to struct *media_device*

struct usb_device * udev pointer to struct usb_device

const char * board_name media device name. If NULL, the routine will use the usb product name, if available.

const char * driver_name name of the driver. if NULL, the routine will use the name given by udev->dev->driver->name, with is usually the wrong thing to do.

Description

Note:

It is better to call media_device_usb_init() *instead, as such macro fills driver_name with KBUILD_MODNAME.*

media_device_usb_init(*mdev*, *udev*, *name*)
create and initialize a struct *media_device* from a PCI device.

Parameters

mdev pointer to struct *media_device*

udev pointer to struct usb_device

name media device name. If NULL, the routine will use the usb product name, if available.

Description

This macro calls *media_device_usb_init()* passing the *media_device_usb_init()* **driver_name** parameter filled with KBUILD_MODNAME.

struct **media_file_operations**
Media device file operations

Definition

```
struct media_file_operations {
  struct module * owner;
  ssize_t (* read) (struct file *, char __user *, size_t, loff_t *);
  ssize_t (* write) (struct file *, const char __user *, size_t, loff_t *);
  unsigned int (* poll) (struct file *, struct poll_table_struct *);
  long (* ioctl) (struct file *, unsigned int, unsigned long);
  long (* compat_ioctl) (struct file *, unsigned int, unsigned long);
  int (* open) (struct file *);
  int (* release) (struct file *);
};
```

Members

owner should be filled with THIS_MODULE

read pointer to the function that implements read() syscall

write pointer to the function that implements *write()* syscall

poll pointer to the function that implements poll() syscall

ioctl pointer to the function that implements ioctl() syscall

compat_ioctl pointer to the function that will handle 32 bits userspace calls to the the ioctl() syscall on a Kernel compiled with 64 bits.

open pointer to the function that implements *open()* syscall

release pointer to the function that will release the resources allocated by the **open** function.

struct **media_devnode**
Media device node

Definition

```
struct media_devnode {
  struct media_device * media_dev;
  const struct media_file_operations * fops;
  struct device dev;
  struct cdev cdev;
  struct device * parent;
  int minor;
  unsigned long flags;
  void (* release) (struct media_devnode *devnode);
};
```

Members

media_dev pointer to struct *media_device*

fops pointer to struct *media_file_operations* with media device ops

dev pointer to struct device containing the media controller device

cdev struct cdev pointer character device

parent parent device

minor device node minor number

flags flags, combination of the MEDIA_FLAG_* constants

release release callback called at the end of :c:func:`media_devnode_release()` routine at media-device.c.

Description

This structure represents a media-related device node.

The **parent** is a physical device. It must be set by core or device drivers before registering the node.

int **media_devnode_register**(struct *media_device* * *mdev*, struct *media_devnode* * *devnode*, struct module * *owner*)
register a media device node

Parameters

struct media_device * mdev struct media_device we want to register a device node

struct media_devnode * devnode media device node structure we want to register

struct module * owner should be filled with THIS_MODULE

Description

The registration code assigns minor numbers and registers the new device node with the kernel. An error is returned if no free minor number can be found, or if the registration of the device node fails.

Zero is returned on success.

Note that if the media_devnode_register call fails, the release() callback of the media_devnode structure is *not* called, so the caller is responsible for freeing any data.

void **media_devnode_unregister_prepare**(struct *media_devnode* * *devnode*)
clear the media device node register bit

Parameters

struct media_devnode * devnode the device node to prepare for unregister

Description

This clears the passed device register bit. Future open calls will be met with errors. Should be called before *media_devnode_unregister()* to avoid races with unregister and device file open calls.

This function can safely be called if the device node has never been registered or has already been unregistered.

void **media_devnode_unregister**(struct *media_devnode* * *devnode*)
unregister a media device node

Parameters

struct media_devnode * devnode the device node to unregister

Description

This unregisters the passed device. Future open calls will be met with errors.

Should be called after *media_devnode_unregister_prepare()*

struct *media_devnode* * **media_devnode_data**(struct file * *filp*)
returns a pointer to the *media_devnode*

Parameters

struct file * filp pointer to struct `file`

int **media_devnode_is_registered**(struct *media_devnode* * *devnode*)
returns true if *media_devnode* is registered; false otherwise.

Parameters

struct media_devnode * devnode pointer to struct *media_devnode*.

Note

If mdev is NULL, it also returns false.

enum **media_gobj_type**
type of a graph object

Constants

MEDIA_GRAPH_ENTITY Identify a media entity

MEDIA_GRAPH_PAD Identify a media pad

MEDIA_GRAPH_LINK Identify a media link

MEDIA_GRAPH_INTF_DEVNODE Identify a media Kernel API interface via a device node

struct **media_gobj**
Define a graph object.

Definition

```
struct media_gobj {
  struct media_device * mdev;
  u32 id;
  struct list_head list;
};
```

Members

mdev Pointer to the struct *media_device* that owns the object

id Non-zero object ID identifier. The ID should be unique inside a media_device, as it is composed by `MEDIA_BITS_PER_TYPE` to store the type plus `MEDIA_BITS_PER_ID` to store the ID

list List entry stored in one of the per-type mdev object lists

Description

All objects on the media graph should have this struct embedded

struct **media_entity_enum**
An enumeration of media entities.

Definition

```
struct media_entity_enum {
  unsigned long * bmap;
  int idx_max;
};
```

Members

bmap Bit map in which each bit represents one entity at struct media_entity->internal_idx.

idx_max Number of bits in bmap

struct **media_graph**
Media graph traversal state

Definition

```
struct media_graph {
  struct stack[MEDIA_ENTITY_ENUM_MAX_DEPTH];
  struct media_entity_enum ent_enum;
  int top;
};
```

Members

stack[MEDIA_ENTITY_ENUM_MAX_DEPTH] Graph traversal stack; the stack contains information on the path the media entities to be walked and the links through which they were reached.

ent_enum Visited entities

top The top of the stack

struct **media_pipeline**
Media pipeline related information

Definition

```
struct media_pipeline {
  int streaming_count;
  struct media_graph graph;
};
```

Members

streaming_count Streaming start count - streaming stop count

graph Media graph walk during pipeline start / stop

struct **media_link**
A link object part of a media graph.

Definition

```
struct media_link {
  struct media_gobj graph_obj;
  struct list_head list;
  union {unnamed_union};
  struct media_link * reverse;
  unsigned long flags;
  bool is_backlink;
};
```

Members

graph_obj Embedded structure containing the media object common data

list Linked list associated with an entity or an interface that owns the link.

{unnamed_union} anonymous

reverse Pointer to the link for the reverse direction of a pad to pad link.

flags Link flags, as defined in uapi/media.h (MEDIA_LNK_FL_*)

is_backlink Indicate if the link is a backlink.

struct **media_pad**
A media pad graph object.

Definition

```
struct media_pad {
  struct media_gobj graph_obj;
  struct media_entity * entity;
  u16 index;
  unsigned long flags;
};
```

Members

graph_obj Embedded structure containing the media object common data

entity Entity this pad belongs to

index Pad index in the entity pads array, numbered from 0 to n

flags Pad flags, as defined in *include/uapi/linux/media.h* (seek for `MEDIA_PAD_FL_*`)

struct **media_entity_operations**
Media entity operations

Definition

```
struct media_entity_operations {
  int (* link_setup) (struct media_entity *entity,const struct media_pad *local,const struct␣
 ↪media_pad *remote, u32 flags);
  int (* link_validate) (struct media_link *link);
};
```

Members

link_setup Notify the entity of link changes. The operation can return an error, in which case link setup will be cancelled. Optional.

link_validate Return whether a link is valid from the entity point of view. The *media_pipeline_start()* function validates all links by calling this operation. Optional.

Description

Note:

Those these callbacks are called with struct `media_device.graph_mutex` *mutex held.*

enum **media_entity_type**
Media entity type

Constants

MEDIA_ENTITY_TYPE_BASE The entity isn't embedded in another subsystem structure.

MEDIA_ENTITY_TYPE_VIDEO_DEVICE The entity is embedded in a struct video_device instance.

MEDIA_ENTITY_TYPE_V4L2_SUBDEV The entity is embedded in a struct v4l2_subdev instance.

Description

Media entity objects are often not instantiated directly, but the media entity structure is inherited by (through embedding) other subsystem-specific structures. The media entity type identifies the type of the subclass structure that implements a media entity instance.

This allows runtime type identification of media entities and safe casting to the correct object type. For instance, a media entity structure instance embedded in a v4l2_subdev structure instance will have the type MEDIA_ENTITY_TYPE_V4L2_SUBDEV and can safely be cast to a *v4l2_subdev* structure using the container_of() macro.

struct **media_entity**
A media entity graph object.

Definition

```
struct media_entity {
  struct media_gobj graph_obj;
  const char * name;
  enum media_entity_type obj_type;
  u32 function;
  unsigned long flags;
  u16 num_pads;
  u16 num_links;
  u16 num_backlinks;
  int internal_idx;
  struct media_pad * pads;
  struct list_head links;
  const struct media_entity_operations * ops;
  int stream_count;
  int use_count;
  struct media_pipeline * pipe;
  union info;
};
```

Members

graph_obj Embedded structure containing the media object common data.

name Entity name.

obj_type Type of the object that implements the media_entity.

function Entity main function, as defined in *include/uapi/linux/media.h* (seek for MEDIA_ENT_F_*)

flags Entity flags, as defined in *include/uapi/linux/media.h* (seek for MEDIA_ENT_FL_*)

num_pads Number of sink and source pads.

num_links Total number of links, forward and back, enabled and disabled.

num_backlinks Number of backlinks

internal_idx An unique internal entity specific number. The numbers are re-used if entities are unregistered or registered again.

pads Pads array with the size defined by **num_pads**.

links List of data links.

ops Entity operations.

stream_count Stream count for the entity.

use_count Use count for the entity.

pipe Pipeline this entity belongs to.

info Union with devnode information. Kept just for backward compatibility.

Description

Note:

stream_count *and* ***use_count*** *reference counts must never be negative, but are signed integers on purpose: a simple* `WARN_ON(<0)` *check can be used to detect reference count bugs that would make them negative.*

struct **media_interface**
A media interface graph object.

Definition

```
struct media_interface {
  struct media_gobj graph_obj;
  struct list_head links;
  u32 type;
  u32 flags;
};
```

Members

graph_obj embedded graph object

links List of links pointing to graph entities

type Type of the interface as defined in *include/uapi/linux/media.h* (seek for `MEDIA_INTF_T_*`)

flags Interface flags as defined in *include/uapi/linux/media.h* (seek for `MEDIA_INTF_FL_*`)

Description

Note:

Currently, no flags for `media_interface` *is defined.*

struct **media_intf_devnode**
A media interface via a device node.

Definition

```
struct media_intf_devnode {
  struct media_interface intf;
  u32 major;
  u32 minor;
};
```

Members

intf embedded interface object

major Major number of a device node

minor Minor number of a device node

u32 **media_entity_id**(struct *media_entity * entity*)
return the media entity graph object id

Parameters

struct media_entity * entity pointer to *media_entity*

enum *media_gobj_type* **media_type**(struct *media_gobj* * *gobj*)
return the media object type

Parameters

struct media_gobj * gobj Pointer to the struct *media_gobj* graph object

u32 **media_id**(struct *media_gobj* * *gobj*)
return the media object ID

Parameters

struct media_gobj * gobj Pointer to the struct *media_gobj* graph object

u32 **media_gobj_gen_id**(enum *media_gobj_type type*, u64 *local_id*)
encapsulates type and ID on at the object ID

Parameters

enum media_gobj_type type object type as define at enum *media_gobj_type*.

u64 local_id next ID, from struct *media_device.id*.

bool **is_media_entity_v4l2_video_device**(struct *media_entity* * *entity*)
Check if the entity is a video_device

Parameters

struct media_entity * entity pointer to entity

Return

`true` if the entity is an instance of a video_device object and can safely be cast to a struct video_device using the `container_of()` macro, or `false` otherwise.

bool **is_media_entity_v4l2_subdev**(struct *media_entity* * *entity*)
Check if the entity is a v4l2_subdev

Parameters

struct media_entity * entity pointer to entity

Return

`true` if the entity is an instance of a *v4l2_subdev* object and can safely be cast to a struct *v4l2_subdev* using the `container_of()` macro, or `false` otherwise.

int **__media_entity_enum_init**(struct *media_entity_enum* * *ent_enum*, int *idx_max*)
Initialise an entity enumeration

Parameters

struct media_entity_enum * ent_enum Entity enumeration to be initialised

int idx_max Maximum number of entities in the enumeration

Return

Returns zero on success or a negative error code.

void **media_entity_enum_cleanup**(struct *media_entity_enum* * *ent_enum*)
Release resources of an entity enumeration

Parameters

struct media_entity_enum * ent_enum Entity enumeration to be released

void **media_entity_enum_zero**(struct *media_entity_enum* * *ent_enum*)
Clear the entire enum

Parameters

struct media_entity_enum * ent_enum Entity enumeration to be cleared

void **media_entity_enum_set**(struct *media_entity_enum * ent_enum*, struct *media_entity * entity*)
Mark a single entity in the enum

Parameters

struct media_entity_enum * ent_enum Entity enumeration

struct media_entity * entity Entity to be marked

void **media_entity_enum_clear**(struct *media_entity_enum * ent_enum*, struct *media_entity * entity*)
Unmark a single entity in the enum

Parameters

struct media_entity_enum * ent_enum Entity enumeration

struct media_entity * entity Entity to be unmarked

bool **media_entity_enum_test**(struct *media_entity_enum * ent_enum*, struct *media_entity * entity*)
Test whether the entity is marked

Parameters

struct media_entity_enum * ent_enum Entity enumeration

struct media_entity * entity Entity to be tested

Description

Returns `true` if the entity was marked.

bool **media_entity_enum_test_and_set**(struct *media_entity_enum * ent_enum*, struct *media_entity * entity*)
Test whether the entity is marked, and mark it

Parameters

struct media_entity_enum * ent_enum Entity enumeration

struct media_entity * entity Entity to be tested

Description

Returns `true` if the entity was marked, and mark it before doing so.

bool **media_entity_enum_empty**(struct *media_entity_enum * ent_enum*)
Test whether the entire enum is empty

Parameters

struct media_entity_enum * ent_enum Entity enumeration

Return

`true` if the entity was empty.

bool **media_entity_enum_intersects**(struct *media_entity_enum * ent_enum1*, struct *media_entity_enum * ent_enum2*)
Test whether two enums intersect

Parameters

struct media_entity_enum * ent_enum1 First entity enumeration

struct media_entity_enum * ent_enum2 Second entity enumeration

Return

`true` if entity enumerations **ent_enum1** and **ent_enum2** intersect, otherwise `false`.

gobj_to_entity(*gobj*)
returns the struct *media_entity* pointer from the **gobj** contained on it.

Parameters

gobj Pointer to the struct *media_gobj* graph object

gobj_to_pad(*gobj*)
returns the struct *media_pad* pointer from the **gobj** contained on it.

Parameters

gobj Pointer to the struct *media_gobj* graph object

gobj_to_link(*gobj*)
returns the struct *media_link* pointer from the **gobj** contained on it.

Parameters

gobj Pointer to the struct *media_gobj* graph object

gobj_to_intf(*gobj*)
returns the struct *media_interface* pointer from the **gobj** contained on it.

Parameters

gobj Pointer to the struct *media_gobj* graph object

intf_to_devnode(*intf*)
returns the struct media_intf_devnode pointer from the **intf** contained on it.

Parameters

intf Pointer to struct *media_intf_devnode*

void **media_gobj_create**(struct *media_device* * *mdev*, enum *media_gobj_type type*, struct *media_gobj* * *gobj*)
Initialize a graph object

Parameters

struct media_device * mdev Pointer to the *media_device* that contains the object

enum media_gobj_type type Type of the object

struct media_gobj * gobj Pointer to the struct *media_gobj* graph object

Description

This routine initializes the embedded struct *media_gobj* inside a media graph object. It is called automatically if `media_*_create` function calls are used. However, if the object (entity, link, pad, interface) is embedded on some other object, this function should be called before registering the object at the media controller.

void **media_gobj_destroy**(struct *media_gobj* * *gobj*)
Stop using a graph object on a media device

Parameters

struct media_gobj * gobj Pointer to the struct *media_gobj* graph object

Description

This should be called by all routines like *media_device_unregister()* that remove/destroy media graph objects.

int **media_entity_pads_init**(struct *media_entity* * *entity*, u16 *num_pads*, struct *media_pad* * *pads*)
Initialize the entity pads

Parameters

struct media_entity * entity entity where the pads belong

u16 num_pads total number of sink and source pads

struct media_pad * pads Array of **num_pads** pads.

Description

The pads array is managed by the entity driver and passed to *media_entity_pads_init()* where its pointer will be stored in the *media_entity* structure.

If no pads are needed, drivers could either directly fill *media_entity->num_pads* with 0 and *media_entity->pads* with NULL or call this function that will do the same.

As the number of pads is known in advance, the pads array is not allocated dynamically but is managed by the entity driver. Most drivers will embed the pads array in a driver-specific structure, avoiding dynamic allocation.

Drivers must set the direction of every pad in the pads array before calling *media_entity_pads_init()*. The function will initialize the other pads fields.

void **media_entity_cleanup**(struct *media_entity * entity*)
free resources associated with an entity

Parameters

struct media_entity * entity entity where the pads belong

Description

This function must be called during the cleanup phase after unregistering the entity (currently, it does nothing).

int **media_create_pad_link**(struct *media_entity * source*, u16 *source_pad*, struct *media_entity * sink*, u16 *sink_pad*, u32 *flags*)
creates a link between two entities.

Parameters

struct media_entity * source pointer to *media_entity* of the source pad.

u16 source_pad number of the source pad in the pads array

struct media_entity * sink pointer to *media_entity* of the sink pad.

u16 sink_pad number of the sink pad in the pads array.

u32 flags Link flags, as defined in *include/uapi/linux/media.h* (seek for MEDIA_LNK_FL_*)

Description

Valid values for flags:

MEDIA_LNK_FL_ENABLED Indicates that the link is enabled and can be used to transfer media data. When two or more links target a sink pad, only one of them can be enabled at a time.

MEDIA_LNK_FL_IMMUTABLE Indicates that the link enabled state can't be modified at runtime. If MEDIA_LNK_FL_IMMUTABLE is set, then MEDIA_LNK_FL_ENABLED must also be set, since an immutable link is always enabled.

Note:

Before calling this function, media_entity_pads_init() *and* media_device_register_entity() *should be called previously for both ends.*

int **media_create_pad_links**(const struct *media_device * mdev*, const u32 *source_function*, struct *media_entity * source*, const u16 *source_pad*, const u32 *sink_function*, struct *media_entity * sink*, const u16 *sink_pad*, u32 *flags*, const bool *allow_both_undefined*)
creates a link between two entities.

Parameters

const struct media_device * mdev Pointer to the media_device that contains the object

const u32 source_function Function of the source entities. Used only if **source** is NULL.

struct media_entity * source pointer to *media_entity* of the source pad. If NULL, it will use all entities that matches the **sink_function**.

const u16 source_pad number of the source pad in the pads array

const u32 sink_function Function of the sink entities. Used only if **sink** is NULL.

struct media_entity * sink pointer to *media_entity* of the sink pad. If NULL, it will use all entities that matches the **sink_function**.

const u16 sink_pad number of the sink pad in the pads array.

u32 flags Link flags, as defined in include/uapi/linux/media.h.

const bool allow_both_undefined if `true`, then both **source** and **sink** can be NULL. In such case, it will create a crossbar between all entities that matches **source_function** to all entities that matches **sink_function**. If `false`, it will return 0 and won't create any link if both **source** and **sink** are NULL.

Description

Valid values for flags:

A `MEDIA_LNK_FL_ENABLED` flag indicates that the link is enabled and can be used to transfer media data. If multiple links are created and this flag is passed as an argument, only the first created link will have this flag.

A `MEDIA_LNK_FL_IMMUTABLE` flag indicates that the link enabled state can't be modified at runtime. If `MEDIA_LNK_FL_IMMUTABLE` is set, then `MEDIA_LNK_FL_ENABLED` must also be set since an immutable link is always enabled.

It is common for some devices to have multiple source and/or sink entities of the same type that should be linked. While *media_create_pad_link()* creates link by link, this function is meant to allow 1:n, n:1 and even cross-bar (n:n) links.

Note:

Before calling this function, `media_entity_pads_init()` *and* `media_device_register_entity()` *should be called previously for the entities to be linked.*

void **media_entity_remove_links**(struct *media_entity* * *entity*)
remove all links associated with an entity

Parameters

struct media_entity * entity pointer to *media_entity*

Description

Note:

This is called automatically when an entity is unregistered via `media_device_register_entity()`.

int **__media_entity_setup_link**(struct *media_link* * *link*, u32 *flags*)
Configure a media link without locking

Parameters

struct media_link * link The link being configured

u32 flags Link configuration flags

Description

The bulk of link setup is handled by the two entities connected through the link. This function notifies both entities of the link configuration change.

If the link is immutable or if the current and new configuration are identical, return immediately.

The user is expected to hold link->source->parent->mutex. If not, *media_entity_setup_link()* should be used instead.

int **media_entity_setup_link**(struct *media_link * link*, u32 *flags*)
changes the link flags properties in runtime

Parameters

struct media_link * link pointer to *media_link*

u32 flags the requested new link flags

Description

The only configurable property is the MEDIA_LNK_FL_ENABLED link flag flag to enable/disable a link. Links marked with the MEDIA_LNK_FL_IMMUTABLE link flag can not be enabled or disabled.

When a link is enabled or disabled, the media framework calls the link_setup operation for the two entities at the source and sink of the link, in that order. If the second link_setup call fails, another link_setup call is made on the first entity to restore the original link flags.

Media device drivers can be notified of link setup operations by setting the *media_device.link_notify* pointer to a callback function. If provided, the notification callback will be called before enabling and after disabling links.

Entity drivers must implement the link_setup operation if any of their links is non-immutable. The operation must either configure the hardware or store the configuration information to be applied later.

Link configuration must not have any side effect on other links. If an enabled link at a sink pad prevents another link at the same pad from being enabled, the link_setup operation must return -EBUSY and can't implicitly disable the first enabled link.

Note:

The valid values of the flags for the link is the same as described on media_create_pad_link(), *for pad to pad links or the same as described on* media_create_intf_link(), *for interface to entity links.*

struct *media_link* * **media_entity_find_link**(struct *media_pad * source*, struct *media_pad * sink*)
Find a link between two pads

Parameters

struct media_pad * source Source pad

struct media_pad * sink Sink pad

Return

returns a pointer to the link between the two entities. If no such link exists, return NULL.

struct *media_pad* * **media_entity_remote_pad**(struct *media_pad * pad*)
Find the pad at the remote end of a link

Parameters

struct media_pad * pad Pad at the local end of the link

Description

Search for a remote pad connected to the given pad by iterating over all links originating or terminating at that pad until an enabled link is found.

Return

returns a pointer to the pad at the remote end of the first found enabled link, or NULL if no enabled link has been found.

struct *media_entity* * **media_entity_get**(struct *media_entity* * *entity*)
Get a reference to the parent module

Parameters

struct media_entity * entity The entity

Description

Get a reference to the parent media device module.

The function will return immediately if **entity** is NULL.

Return

returns a pointer to the entity on success or NULL on failure.

int **media_graph_walk_init**(struct *media_graph* * *graph*, struct *media_device* * *mdev*)
Allocate resources used by graph walk.

Parameters

struct media_graph * graph Media graph structure that will be used to walk the graph

struct media_device * mdev Pointer to the *media_device* that contains the object

void **media_graph_walk_cleanup**(struct *media_graph* * *graph*)
Release resources used by graph walk.

Parameters

struct media_graph * graph Media graph structure that will be used to walk the graph

void **media_entity_put**(struct *media_entity* * *entity*)
Release the reference to the parent module

Parameters

struct media_entity * entity The entity

Description

Release the reference count acquired by *media_entity_get()*.

The function will return immediately if **entity** is NULL.

void **media_graph_walk_start**(struct *media_graph* * *graph*, struct *media_entity* * *entity*)
Start walking the media graph at a given entity

Parameters

struct media_graph * graph Media graph structure that will be used to walk the graph

struct media_entity * entity Starting entity

Description

Before using this function, *media_graph_walk_init()* must be used to allocate resources used for walking the graph. This function initializes the graph traversal structure to walk the entities graph starting at the given entity. The traversal structure must not be modified by the caller during graph traversal. After the graph walk, the resources must be released using *media_graph_walk_cleanup()*.

struct *media_entity* * **media_graph_walk_next**(struct *media_graph* * *graph*)
Get the next entity in the graph

Parameters

struct media_graph * graph Media graph structure

Description

Perform a depth-first traversal of the given media entities graph.

The graph structure must have been previously initialized with a call to *media_graph_walk_start()*.

Return

returns the next entity in the graph or NULL if the whole graph have been traversed.

int **media_pipeline_start**(struct *media_entity* * *entity*, struct *media_pipeline* * *pipe*)
Mark a pipeline as streaming

Parameters

struct media_entity * entity Starting entity

struct media_pipeline * pipe Media pipeline to be assigned to all entities in the pipeline.

Description

Mark all entities connected to a given entity through enabled links, either directly or indirectly, as streaming. The given pipeline object is assigned to every entity in the pipeline and stored in the media_entity pipe field.

Calls to this function can be nested, in which case the same number of *media_pipeline_stop()* calls will be required to stop streaming. The pipeline pointer must be identical for all nested calls to *media_pipeline_start()*.

int **__media_pipeline_start**(struct *media_entity* * *entity*, struct *media_pipeline* * *pipe*)
Mark a pipeline as streaming

Parameters

struct media_entity * entity Starting entity

struct media_pipeline * pipe Media pipeline to be assigned to all entities in the pipeline.

Description

..note:: This is the non-locking version of *media_pipeline_start()*

void **media_pipeline_stop**(struct *media_entity* * *entity*)
Mark a pipeline as not streaming

Parameters

struct media_entity * entity Starting entity

Description

Mark all entities connected to a given entity through enabled links, either directly or indirectly, as not streaming. The media_entity pipe field is reset to NULL.

If multiple calls to *media_pipeline_start()* have been made, the same number of calls to this function are required to mark the pipeline as not streaming.

void **__media_pipeline_stop**(struct *media_entity* * *entity*)
Mark a pipeline as not streaming

Parameters

struct media_entity * entity Starting entity

Description

Note:

This is the non-locking version of media_pipeline_stop()

struct *media_intf_devnode* * **media_devnode_create**(struct *media_device* * *mdev*, u32 *type*, u32 *flags*, u32 *major*, u32 *minor*)
creates and initializes a device node interface

Parameters

struct media_device * mdev pointer to struct *media_device*

u32 type type of the interface, as given by *include/uapi/linux/media.h* (seek for MEDIA_INTF_T_*) macros.

u32 flags Interface flags, as defined in *include/uapi/linux/media.h* (seek for MEDIA_INTF_FL_*)

u32 major Device node major number.

u32 minor Device node minor number.

Return

if succeeded, returns a pointer to the newly allocated *media_intf_devnode* pointer.

Note:

Currently, no flags for media_interface *is defined.*

void **media_devnode_remove**(struct *media_intf_devnode* * *devnode*)
removes a device node interface

Parameters

struct media_intf_devnode * devnode pointer to *media_intf_devnode* to be freed.

Description

When a device node interface is removed, all links to it are automatically removed.

media_create_intf_link(struct *media_entity* * *entity*, struct *media_interface* * *intf*, u32 *flags*)
creates a link between an entity and an interface

Parameters

struct media_entity * entity pointer to media_entity

struct media_interface * intf pointer to media_interface

u32 flags Link flags, as defined in *include/uapi/linux/media.h* (seek for MEDIA_LNK_FL_*)

Description

Valid values for flags:

MEDIA_LNK_FL_ENABLED Indicates that the interface is connected to the entity hardware. That's the default value for interfaces. An interface may be disabled if the hardware is busy due to the usage of some other interface that it is currently controlling the hardware.

A typical example is an hybrid TV device that handle only one type of stream on a given time. So, when the digital TV is streaming, the V4L2 interfaces won't be enabled, as such device is not able to also stream analog TV or radio.

Note:

Before calling this function, media_devnode_create() *should be called for the interface and* media_device_register_entity() *should be called for the interface that will be part of the link.*

void **__media_remove_intf_link**(struct *media_link* * *link*)
remove a single interface link

Parameters

struct media_link * link pointer to *media_link*.

Description

Note:

This is an unlocked version of media_remove_intf_link()

void **media_remove_intf_link**(struct *media_link* * *link*)
remove a single interface link

Parameters

struct media_link * link pointer to *media_link*.

Description

Note:

Prefer to use this one, instead of __media_remove_intf_link()

void **__media_remove_intf_links**(struct *media_interface* * *intf*)
remove all links associated with an interface

Parameters

struct media_interface * intf pointer to *media_interface*

Description

Note:

This is an unlocked version of media_remove_intf_links().

void **media_remove_intf_links**(struct *media_interface* * *intf*)
remove all links associated with an interface

Parameters

struct media_interface * intf pointer to *media_interface*

Description

Note:

1. *This is called automatically when an entity is unregistered via* media_device_register_entity() *and by* media_devnode_remove().
2. *Prefer to use this one, instead of* __media_remove_intf_links().

media_entity_call(*entity*, *operation*, *args...*)
Calls a struct media_entity_operations operation on an entity

Parameters

entity entity where the **operation** will be called

operation type of the operation. Should be the name of a member of struct *media_entity_operations*.

args... variable arguments

Description

This helper function will check if **operation** is not NULL. On such case, it will issue a call to **operation**(**entity**, **args**).

2.11 CEC Kernel Support

The CEC framework provides a unified kernel interface for use with HDMI CEC hardware. It is designed to handle a multiple types of hardware (receivers, transmitters, USB dongles). The framework also gives the option to decide what to do in the kernel driver and what should be handled by userspace applications. In addition it integrates the remote control passthrough feature into the kernel's remote control framework.

2.11.1 The CEC Protocol

The CEC protocol enables consumer electronic devices to communicate with each other through the HDMI connection. The protocol uses logical addresses in the communication. The logical address is strictly connected with the functionality provided by the device. The TV acting as the communication hub is always assigned address 0. The physical address is determined by the physical connection between devices.

The CEC framework described here is up to date with the CEC 2.0 specification. It is documented in the HDMI 1.4 specification with the new 2.0 bits documented in the HDMI 2.0 specification. But for most of the features the freely available HDMI 1.3a specification is sufficient:

http://www.microprocessor.org/HDMISpecification13a.pdf

2.12 The Kernel Interface

2.12.1 CEC Adapter

The struct cec_adapter represents the CEC adapter hardware. It is created by calling cec_allocate_adapter() and deleted by calling cec_delete_adapter():

```
struct cec_adapter *cec_allocate_adapter(const struct cec_adap_ops *ops, void *priv,
const char *name, u32 caps, u8 available_las);
```

```
void cec_delete_adapter(struct cec_adapter *adap);
```

To create an adapter you need to pass the following information:

ops: adapter operations which are called by the CEC framework and that you have to implement.

priv: will be stored in adap->priv and can be used by the adapter ops.

name: the name of the CEC adapter. Note: this name will be copied.

caps: capabilities of the CEC adapter. These capabilities determine the capabilities of the hardware and which parts are to be handled by userspace and which parts are handled by kernelspace. The capabilities are returned by CEC_ADAP_G_CAPS.

available_las: the number of simultaneous logical addresses that this adapter can handle. Must be 1 <= available_las <= CEC_MAX_LOG_ADDRS.

To register the /dev/cecX device node and the remote control device (if CEC_CAP_RC is set) you call:

```
int cec_register_adapter(struct cec_adapter *adap, struct device *parent);
```

where parent is the parent device.

To unregister the devices call:

```
void cec_unregister_adapter(struct cec_adapter *adap);
```

Note: if cec_register_adapter() fails, then call cec_delete_adapter() to clean up. But if cec_register_adapter() succeeded, then only call cec_unregister_adapter() to clean up, never cec_delete_adapter(). The unregister function will delete the adapter automatically once the last user of that /dev/cecX device has closed its file handle.

2.12.2 Implementing the Low-Level CEC Adapter

The following low-level adapter operations have to be implemented in your driver:

struct **cec_adap_ops**

```
struct cec_adap_ops
{
        /* Low-level callbacks */
        int (*adap_enable)(struct cec_adapter *adap, bool enable);
        int (*adap_monitor_all_enable)(struct cec_adapter *adap, bool enable);
        int (*adap_log_addr)(struct cec_adapter *adap, u8 logical_addr);
        int (*adap_transmit)(struct cec_adapter *adap, u8 attempts,
                             u32 signal_free_time, struct cec_msg *msg);
        void (*adap_status)(struct cec_adapter *adap, struct seq_file *file);

        /* High-level callbacks */
        ...
};
```

The five low-level ops deal with various aspects of controlling the CEC adapter hardware:

To enable/disable the hardware:

int (*adap_enable)(struct cec_adapter *adap, bool enable);

This callback enables or disables the CEC hardware. Enabling the CEC hardware means powering it up in a state where no logical addresses are claimed. This op assumes that the physical address (adap->phys_addr) is valid when enable is true and will not change while the CEC adapter remains enabled. The initial state of the CEC adapter after calling cec_allocate_adapter() is disabled.

Note that adap_enable must return 0 if enable is false.

To enable/disable the 'monitor all' mode:

int (*adap_monitor_all_enable)(struct cec_adapter *adap, bool enable);

If enabled, then the adapter should be put in a mode to also monitor messages that not for us. Not all hardware supports this and this function is only called if the CEC_CAP_MONITOR_ALL capability is set. This callback is optional (some hardware may always be in 'monitor all' mode).

Note that adap_monitor_all_enable must return 0 if enable is false.

To program a new logical address:

int (*adap_log_addr)(struct cec_adapter *adap, u8 logical_addr);

If logical_addr == CEC_LOG_ADDR_INVALID then all programmed logical addresses are to be erased. Otherwise the given logical address should be programmed. If the maximum number of available logical addresses is exceeded, then it should return -ENXIO. Once a logical address is programmed the CEC hardware can receive directed messages to that address.

Note that adap_log_addr must return 0 if logical_addr is CEC_LOG_ADDR_INVALID.

To transmit a new message:

int (*adap_transmit)(struct cec_adapter *adap, u8 attempts,
u32 signal_free_time, struct cec_msg *msg);

This transmits a new message. The attempts argument is the suggested number of attempts for the transmit.

The signal_free_time is the number of data bit periods that the adapter should wait when the line is free before attempting to send a message. This value depends on whether this transmit is a retry, a message from a new initiator or a new message for the same initiator. Most hardware will handle this automatically, but in some cases this information is needed.

The CEC_FREE_TIME_TO_USEC macro can be used to convert signal_free_time to microseconds (one data bit period is 2.4 ms).

To log the current CEC hardware status:

```
void (*adap_status)(struct cec_adapter *adap, struct seq_file *file);
```

This optional callback can be used to show the status of the CEC hardware. The status is available through debugfs: cat /sys/kernel/debug/cec/cecX/status

Your adapter driver will also have to react to events (typically interrupt driven) by calling into the framework in the following situations:

When a transmit finished (successfully or otherwise):

```
void cec_transmit_done(struct cec_adapter *adap, u8 status, u8 arb_lost_cnt,
u8 nack_cnt, u8 low_drive_cnt, u8 error_cnt);
```

The status can be one of:

CEC_TX_STATUS_OK: the transmit was successful.

CEC_TX_STATUS_ARB_LOST: arbitration was lost: another CEC initiator took control of the CEC line and you lost the arbitration.

CEC_TX_STATUS_NACK: the message was nacked (for a directed message) or acked (for a broadcast message). A retransmission is needed.

CEC_TX_STATUS_LOW_DRIVE: low drive was detected on the CEC bus. This indicates that a follower detected an error on the bus and requested a retransmission.

CEC_TX_STATUS_ERROR: some unspecified error occurred: this can be one of the previous two if the hardware cannot differentiate or something else entirely.

CEC_TX_STATUS_MAX_RETRIES: could not transmit the message after trying multiple times. Should only be set by the driver if it has hardware support for retrying messages. If set, then the framework assumes that it doesn't have to make another attempt to transmit the message since the hardware did that already.

The *_cnt arguments are the number of error conditions that were seen. This may be 0 if no information is available. Drivers that do not support hardware retry can just set the counter corresponding to the transmit error to 1, if the hardware does support retry then either set these counters to 0 if the hardware provides no feedback of which errors occurred and how many times, or fill in the correct values as reported by the hardware.

When a CEC message was received:

```
void cec_received_msg(struct cec_adapter *adap, struct cec_msg *msg);
```

Speaks for itself.

2.12.3 Implementing the interrupt handler

Typically the CEC hardware provides interrupts that signal when a transmit finished and whether it was successful or not, and it provides and interrupt when a CEC message was received.

The CEC driver should always process the transmit interrupts first before handling the receive interrupt. The framework expects to see the cec_transmit_done call before the cec_received_msg call, otherwise it can get confused if the received message was in reply to the transmitted message.

2.12.4 Implementing the High-Level CEC Adapter

The low-level operations drive the hardware, the high-level operations are CEC protocol driven. The following high-level callbacks are available:

```
struct cec_adap_ops {
        /* Low-level callbacks */
        ...

        /* High-level CEC message callback */
        int (*received)(struct cec_adapter *adap, struct cec_msg *msg);
};
```

The received() callback allows the driver to optionally handle a newly received CEC message

int (*received)(struct cec_adapter *adap, struct cec_msg *msg);

If the driver wants to process a CEC message, then it can implement this callback. If it doesn't want to handle this message, then it should return -ENOMSG, otherwise the CEC framework assumes it processed this message and it will not do anything with it.

2.12.5 CEC framework functions

CEC Adapter drivers can call the following CEC framework functions:

**int cec_transmit_msg(struct cec_adapter *adap, struct cec_msg *msg,
bool block);**

Transmit a CEC message. If block is true, then wait until the message has been transmitted, otherwise just queue it and return.

**void cec_s_phys_addr(struct cec_adapter *adap, u16 phys_addr,
bool block);**

Change the physical address. This function will set adap->phys_addr and send an event if it has changed. If cec_s_log_addrs() has been called and the physical address has become valid, then the CEC framework will start claiming the logical addresses. If block is true, then this function won't return until this process has finished.

When the physical address is set to a valid value the CEC adapter will be enabled (see the adap_enable op). When it is set to CEC_PHYS_ADDR_INVALID, then the CEC adapter will be disabled. If you change a valid physical address to another valid physical address, then this function will first set the address to CEC_PHYS_ADDR_INVALID before enabling the new physical address.

**int cec_s_log_addrs(struct cec_adapter *adap,
struct cec_log_addrs *log_addrs, bool block);**

Claim the CEC logical addresses. Should never be called if CEC_CAP_LOG_ADDRS is set. If block is true, then wait until the logical addresses have been claimed, otherwise just queue it and return. To unconfigure all logical addresses call this function with log_addrs set to NULL or with log_addrs->num_log_addrs set to 0. The block argument is ignored when unconfiguring. This function will just return if the physical address is invalid. Once the physical address becomes valid, then the framework will attempt to claim these logical addresses.

2.13 MIPI CSI-2

CSI-2 is a data bus intended for transferring images from cameras to the host SoC. It is defined by the MIPI alliance.

2.13.1 Transmitter drivers

CSI-2 transmitter, such as a sensor or a TV tuner, drivers need to provide the CSI-2 receiver with information on the CSI-2 bus configuration. These include the V4L2_CID_LINK_FREQ and V4L2_CID_PIXEL_RATE controls and (*v4l2_subdev_video_ops*->s_stream() callback). These interface elements must be present on the sub-device represents the CSI-2 transmitter.

The V4L2_CID_LINK_FREQ control is used to tell the receiver driver the frequency (and not the symbol rate) of the link. The V4L2_CID_PIXEL_RATE is may be used by the receiver to obtain the pixel rate the transmitter uses. The *v4l2_subdev_video_ops*->s_stream() callback provides an ability to start and stop the stream.

The value of the V4L2_CID_PIXEL_RATE is calculated as follows:

```
pixel_rate = link_freq * 2 * nr_of_lanes / bits_per_sample
```

where

Table 2.1: variables in pixel rate calculation

variable or constant	description
link_freq	The value of the V4L2_CID_LINK_FREQ integer64 menu item.
nr_of_lanes	Number of data lanes used on the CSI-2 link. This can be obtained from the OF endpoint configuration.
2	Two bits are transferred per clock cycle per lane.
bits_per_sample	Number of bits per sample.

The transmitter drivers must configure the CSI-2 transmitter to *LP-11 mode* whenever the transmitter is powered on but not active. Some transmitters do this automatically but some have to be explicitly programmed to do so.

2.13.2 Receiver drivers

Before the receiver driver may enable the CSI-2 transmitter by using the *v4l2_subdev_video_ops*->s_stream(), it must have powered the transmitter up by using the *v4l2_subdev_core_ops*->s_power() callback. This may take place either indirectly by using *v4l2_pipeline_pm_use()* or directly.

CHAPTER

THREE

LINUX DIGITAL TV DRIVER-SPECIFIC DOCUMENTATION

Table of Contents

3.1 Introduction

The main development site and GIT repository for these drivers is https://linuxtv.org.

The DVB mailing list linux-dvb is hosted at vger. Please see http://vger.kernel.org/vger-lists.html#linux-media for details.

There are also some other old lists hosted at https://linuxtv.org/lists.php. Please check the archive https://linuxtv.org/pipermail/linux-dvb/.

The media subsystem Wiki is hosted at https://linuxtv.org/wiki/. Please check it before asking newbie questions on the list.

API documentation is documented at the Kernel. You'll also find useful documentation at: https://linuxtv.org/docs.php.

You may also find useful material at https://linuxtv.org/downloads/.

In order to get firmware from proprietary drivers, there's a script at the kernel tree, at scripts/get_dvb_firmware.

3.2 HOWTO: Get An Avermedia DVB-T working under Linux

February 14th 2006

Note:

This documentation is outdated. Please check at the DVB wiki at https://linuxtv.org/wiki for more updated info.
There's a section there specific for Avermedia boards at: https://linuxtv.org/wiki/index.php/AVerMedia

3.2.1 Assumptions and Introduction

It is assumed that the reader understands the basic structure of the Linux Kernel DVB drivers and the general principles of Digital TV.

One significant difference between Digital TV and Analogue TV that the unwary (like myself) should consider is that, although the component structure of budget DVB-T cards are substantially similar to Analogue TV cards, they function in substantially different ways.

The purpose of an Analogue TV is to receive and display an Analogue Television signal. An Analogue TV signal (otherwise known as composite video) is an analogue encoding of a sequence of image frames (25 per second) rasterised using an interlacing technique. Interlacing takes two fields to represent one frame. Computers today are at their best when dealing with digital signals, not analogue signals and a composite video signal is about as far removed from a digital data stream as you can get. Therefore, an Analogue TV card for a PC has the following purpose:

- Tune the receiver to receive a broadcast signal
- demodulate the broadcast signal
- demultiplex the analogue video signal and analogue audio signal. **NOTE:** some countries employ a digital audio signal embedded within the modulated composite analogue signal - NICAM.)
- digitize the analogue video signal and make the resulting datastream available to the data bus.

The digital datastream from an Analogue TV card is generated by circuitry on the card and is often presented uncompressed. For a PAL TV signal encoded at a resolution of 768x576 24-bit color pixels over 25 frames per second - a fair amount of data is generated and must be processed by the PC before it can be displayed on the video monitor screen. Some Analogue TV cards for PCs have onboard MPEG2 encoders which permit the raw digital data stream to be presented to the PC in an encoded and compressed form - similar to the form that is used in Digital TV.

The purpose of a simple budget digital TV card (DVB-T,C or S) is to simply:

- Tune the received to receive a broadcast signal.
- Extract the encoded digital datastream from the broadcast signal.
- Make the encoded digital datastream (MPEG2) available to the data bus.

The significant difference between the two is that the tuner on the analogue TV card spits out an Analogue signal, whereas the tuner on the digital TV card spits out a compressed encoded digital datastream. As the signal is already digitised, it is trivial to pass this datastream to the PC databus with minimal additional processing and then extract the digital video and audio datastreams passing them to the appropriate software or hardware for decoding and viewing.

3.2.2 The Avermedia DVB-T

The Avermedia DVB-T is a budget PCI DVB card. It has 3 inputs:

- RF Tuner Input
- Composite Video Input (RCA Jack)
- SVIDEO Input (Mini-DIN)

The RF Tuner Input is the input to the tuner module of the card. The Tuner is otherwise known as the “Frontend” . The Frontend of the Avermedia DVB-T is a Microtune 7202D. A timely post to the linux-dvb mailing list ascertained that the Microtune 7202D is supported by the sp887x driver which is found in the dvb-hw CVS module.

The DVB-T card is based around the BT878 chip which is a very common multimedia bridge and often found on Analogue TV cards. There is no on-board MPEG2 decoder, which means that all MPEG2 decoding must be done in software, or if you have one, on an MPEG2 hardware decoding card or chipset.

3.2.3 Getting the card going

In order to fire up the card, it is necessary to load a number of modules from the DVB driver set. Prior to this it will have been necessary to download these drivers from the linuxtv CVS server and compile them successfully.

Depending on the card's feature set, the Device Driver API for DVB under Linux will expose some of the following device files in the /dev tree:

- /dev/dvb/adapter0/audio0
- /dev/dvb/adapter0/ca0
- /dev/dvb/adapter0/demux0
- /dev/dvb/adapter0/dvr0
- /dev/dvb/adapter0/frontend0
- /dev/dvb/adapter0/net0
- /dev/dvb/adapter0/osd0
- /dev/dvb/adapter0/video0

The primary device nodes that we are interested in (at this stage) for the Avermedia DVB-T are:

- /dev/dvb/adapter0/dvr0
- /dev/dvb/adapter0/frontend0

The dvr0 device node is used to read the MPEG2 Data Stream and the frontend0 node is used to tune the frontend tuner module.

At this stage, it has not been able to ascertain the functionality of the remaining device nodes in respect of the Avermedia DVBT. However, full functionality in respect of tuning, receiving and supplying the MPEG2 data stream is possible with the currently available versions of the driver. It may be possible that additional functionality is available from the card (i.e. viewing the additional analogue inputs that the card presents), but this has not been tested yet. If I get around to this, I'll update the document with whatever I find.

To power up the card, load the following modules in the following order:

- modprobe bttv (normally loaded automatically)
- modprobe dvb-bt8xx (or place dvb-bt8xx in /etc/modules)

Insertion of these modules into the running kernel will activate the appropriate DVB device nodes. It is then possible to start accessing the card with utilities such as scan, tzap, dvbstream etc.

The frontend module sp887x.o, requires an external firmware. Please use the command "get_dvb_firmware sp887x" to download it. Then copy it to /usr/lib/hotplug/firmware or /lib/firmware/ (depending on configuration of firmware hotplug).

3.2.4 Receiving DVB-T in Australia

I have no experience of DVB-T in other countries other than Australia, so I will attempt to explain how it works here in Melbourne and how this affects the configuration of the DVB-T card.

The Digital Broadcasting Australia website has a Reception locatortool which provides information on transponder channels and frequencies. My local transmitter happens to be Mount Dandenong.

The frequencies broadcast by Mount Dandenong are:

Table 1. Transponder Frequencies Mount Dandenong, Vic, Aus. Broadcaster Channel Frequency ABC VHF 12 226.5 MHz TEN VHF 11 219.5 MHz NINE VHF 8 191.625 MHz SEVEN VHF 6 177.5 MHz SBS UHF 29 536.5 MHz

The Scan utility has a set of compiled-in defaults for various countries and regions, but if they do not suit, or if you have a pre-compiled scan binary, you can specify a data file on the command line which contains the transponder frequencies. Here is a sample file for the above channel transponders:

```
# Data file for DVB scan program
#
# C Frequency SymbolRate FEC QAM
# S Frequency Polarisation SymbolRate FEC
# T Frequency Bandwidth FEC FEC2 QAM Mode Guard Hier
T 226500000 7MHz 2/3 NONE QAM64 8k 1/8 NONE
T 191625000 7MHz 2/3 NONE QAM64 8k 1/8 NONE
T 219500000 7MHz 2/3 NONE QAM64 8k 1/8 NONE
T 177500000 7MHz 2/3 NONE QAM64 8k 1/8 NONE
T 536500000 7MHz 2/3 NONE QAM64 8k 1/8 NONE
```

The defaults for the transponder frequency and other modulation parameters were obtained from www.dba.org.au.

When Scan runs, it will output channels.conf information for any channel's transponders which the card's frontend can lock onto. (i.e. any whose signal is strong enough at your antenna).

Here's my channels.conf file for anyone who's interested:

```
ABC HDTV:226500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_3_4:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:2307:0:560
ABC TV Melbourne:226500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_3_4:QAM_64:TRANSMISSION_
  ↪MODE_8K:GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:561
ABC TV 2:226500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_3_4:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:562
ABC TV 3:226500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_3_4:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:563
ABC TV 4:226500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_3_4:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:564
ABC DiG Radio:226500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_3_4:QAM_64:TRANSMISSION_MODE_
  ↪8K:GUARD_INTERVAL_1_16:HIERARCHY_NONE:0:2311:566
TEN Digital:219500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:1585
TEN Digital 1:219500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_
  ↪8K:GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:1586
TEN Digital 2:219500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_
  ↪8K:GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:1587
TEN Digital 3:219500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_
  ↪8K:GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:1588
TEN Digital:219500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:1589
TEN Digital 4:219500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_
  ↪8K:GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:1590
TEN Digital:219500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:1591
TEN HD:219500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:514:0:1592
TEN Digital:219500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:650:1593
Nine Digital:191625000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_
  ↪8K:GUARD_INTERVAL_1_16:HIERARCHY_NONE:513:660:1072
Nine Digital HD:191625000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_
  ↪MODE_8K:GUARD_INTERVAL_1_16:HIERARCHY_NONE:512:0:1073
Nine Guide:191625000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_3_4:FEC_1_2:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_16:HIERARCHY_NONE:514:670:1074
7 Digital:177500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_8:HIERARCHY_NONE:769:770:1328
7 Digital 1:177500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_8:HIERARCHY_NONE:769:770:1329
7 Digital 2:177500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_8K:
  ↪GUARD_INTERVAL_1_8:HIERARCHY_NONE:769:770:1330
```

```
7 Digital 3:177500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_8K:
 ↪GUARD_INTERVAL_1_8:HIERARCHY_NONE:769:770:1331
7 HD Digital:177500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_
 ↪8K:GUARD_INTERVAL_1_8:HIERARCHY_NONE:833:834:1332
7 Program Guide:177500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_
 ↪MODE_8K:GUARD_INTERVAL_1_8:HIERARCHY_NONE:865:866:1334
SBS HD:536500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_8K:
 ↪GUARD_INTERVAL_1_8:HIERARCHY_NONE:102:103:784
SBS DIGITAL 1:536500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_
 ↪8K:GUARD_INTERVAL_1_8:HIERARCHY_NONE:161:81:785
SBS DIGITAL 2:536500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_
 ↪8K:GUARD_INTERVAL_1_8:HIERARCHY_NONE:162:83:786
SBS EPG:536500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_8K:
 ↪GUARD_INTERVAL_1_8:HIERARCHY_NONE:163:85:787
SBS RADIO 1:536500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_8K:
 ↪GUARD_INTERVAL_1_8:HIERARCHY_NONE:0:201:798
SBS RADIO 2:536500000:INVERSION_OFF:BANDWIDTH_7_MHZ:FEC_2_3:FEC_2_3:QAM_64:TRANSMISSION_MODE_8K:
 ↪GUARD_INTERVAL_1_8:HIERARCHY_NONE:0:202:799
```

3.2.5 Known Limitations

At present I can say with confidence that the frontend tunes via /dev/dvb/adapter{x}/frontend0 and supplies an MPEG2 stream via /dev/dvb/adapter{x}/dvr0. I have not tested the functionality of any other part of the card yet. I will do so over time and update this document.

There are some limitations in the i2c layer due to a returned error message inconsistency. Although this generates errors in dmesg and the system logs, it does not appear to affect the ability of the frontend to function correctly.

3.2.6 Further Update

dvbstream and VideoLAN Client on windows works a treat with DVB, in fact this is currently serving as my main way of viewing DVB-T at the moment. Additionally, VLC is happily decoding HDTV signals, although the PC is dropping the odd frame here and there - I assume due to processing capability - as all the decoding is being done under windows in software.

Many thanks to Nigel Pearson for the updates to this document since the recent revision of the driver.

3.3 How to get the bt8xx cards working

Authors: Richard Walker, Jamie Honan, Michael Hunold, Manu Abraham, Uwe Bugla, Michael Krufky

Note:

This documentation is outdated. Please check at the DVB wiki at https://linuxtv.org/wiki for more updated info.

3.3.1 General information

This class of cards has a bt878a as the PCI interface, and require the bttv driver for accessing the i2c bus and the gpio pins of the bt8xx chipset. Please see Documentation/dvb/cards.txt => o Cards based on the Conexant Bt8xx PCI bridge:

Compiling kernel please enable:

1. Device drivers => Multimedia devices => Video For Linux => Enable Video for Linux API 1 (DEPRECATED)
2. Device drivers => Multimedia devices => Video For Linux => Video Capture Adapters => BT848 Video For Linux
3. Device drivers => Multimedia devices => Digital Video Broadcasting Devices => DVB for Linux DVB Core Support Bt8xx based PCI Cards

   Please use the following options with care as deselection of drivers which are in fact necessary may result in DVB devices that cannot be tuned due to lack of driver support: You can save RAM by deselecting every frontend module that your DVB card does not need.

   First please remove the static dependency of DVB card drivers on all frontend modules for all possible card variants by enabling:

1. Device drivers => Multimedia devices => Digital Video Broadcasting Devices => DVB for Linux DVB Core Support Load and attach frontend modules as needed

   If you know the frontend driver that your card needs please enable:

1. Device drivers => Multimedia devices => Digital Video Broadcasting Devices => DVB for Linux DVB Core Support Customise DVB Frontends => Customise the frontend modules to build

   Then please select your card-specific frontend module.

3.3.2 Loading Modules

Regular case: If the bttv driver detects a bt8xx-based DVB card, all frontend and backend modules will be loaded automatically. Exceptions are: - Old TwinHan DST cards or clones with or without CA slot and not containing an Eeprom. People running udev please see Documentation/dvb/udev.txt.

In the following cases overriding the PCI type detection for dvb-bt8xx might be necessary:

Running TwinHan and Clones

```
$ modprobe bttv card=113
$ modprobe dst
```

Useful parameters for verbosity level and debugging the dst module:

```
verbose=0:              messages are disabled
        1:              only error messages are displayed
        2:              notifications are displayed
        3:              other useful messages are displayed
        4:              debug setting
dst_addons=0:           card is a free to air (FTA) card only
        0x20:   card has a conditional access slot for scrambled channels
```

The autodetected values are determined by the cards' "response string". In your logs see f. ex.: dst_get_device_id: Recognize [DSTMCI]. For bug reports please send in a complete log with verbose=4 activated. Please also see Documentation/dvb/ci.txt.

Running multiple cards

Examples of card ID's:

```
Pinnacle PCTV Sat:               94
Nebula Electronics Digi TV:     104
pcHDTV HD-2000 TV:              112
```

```
Twinhan DST and clones:          113
Avermedia AverTV DVB-T 771:      123
Avermedia AverTV DVB-T 761:      124
DViCO FusionHDTV DVB-T Lite:     128
DViCO FusionHDTV 5 Lite:         135
```

Note:

The order of the card ID should be uprising:
Example:

```
$ modprobe bttv card=113 card=135
```

For a full list of card ID's please see Documentation/video4linux/CARDLIST.bttv. In case of further problems please subscribe and send questions to the mailing list: linux-dvb@linuxtv.org.

Probing the cards with broken PCI subsystem ID

There are some TwinHan cards that the EEPROM has become corrupted for some reason. The cards do not have correct PCI subsystem ID. But we can force probing the cards with broken PCI subsystem ID

```
$ echo 109e 0878 $subvendor $subdevice > \
       /sys/bus/pci/drivers/bt878/new_id
```

```
109e: PCI_VENDOR_ID_BROOKTREE
0878: PCI_DEVICE_ID_BROOKTREE_878
```

3.4 Hardware supported by the linuxtv.org DVB drivers

Note:

This documentation is outdated. Please check at the DVB wiki at https://linuxtv.org/wiki for more updated info.
Please look at https://linuxtv.org/wiki/index.php/Hardware_Device_Information for an updated list of supported cards.

Generally, the DVB hardware manufacturers frequently change the frontends (i.e. tuner / demodulator units) used, usually without changing the product name, revision number or specs. Some cards are also available in versions with different frontends for DVB-S/DVB-C/DVB-T. Thus the frontend drivers are listed separately.

Note:

1. *There is no guarantee that every frontend driver works out of the box with every card, because of different wiring.*
2. *The demodulator chips can be used with a variety of tuner/PLL chips, and not all combinations are supported. Often the demodulator and tuner/PLL chip are inside a metal box for shielding, and the whole metal box has its own part number.*

- Frontends drivers:

- dvb_dummy_fe: for testing...

DVB-S:

- ves1x93 : Alps BSRV2 (ves1893 demodulator) and dbox2 (ves1993)
- cx24110 : Conexant HM1221/HM1811 (cx24110 or cx24106 demod, cx24108 PLL)
- grundig_29504-491 : Grundig 29504-491 (Philips TDA8083 demodulator), tsa5522 PLL
- mt312 : Zarlink mt312 or Mitel vp310 demodulator, sl1935 or tsa5059 PLLi, Technisat Sky2Pc with bios Rev. 2.3
- **stv0299** [Alps BSRU6 (tsa5059 PLL), LG TDQB-S00x (tsa5059 PLL),] LG TDQF-S001F (sl1935 PLL), Philips SU1278 (tua6100 PLL), Philips SU1278SH (tsa5059 PLL), Samsung TBMU24112IMB, Technisat Sky2Pc with bios Rev. 2.6

DVB-C:

- ves1820 : various (ves1820 demodulator, sp5659c or spXXXX PLL)
- at76c651 : Atmel AT76c651(B) with DAT7021 PLL

DVB-T:

- alps_tdlb7 : Alps TDLB7 (sp8870 demodulator, sp5659 PLL)
- alps_tdmb7 : Alps TDMB7 (cx22700 demodulator)
- grundig_29504-401 : Grundig 29504-401 (LSI L64781 demodulator), tsa5060 PLL
- tda1004x : Philips tda10045h (td1344 or tdm1316l PLL)
- nxt6000 : Alps TDME7 (MITEL SP5659 PLL), Alps TDED4 (TI ALP510 PLL), Comtech DVBT-6k07 (SP5730 PLL), (NxtWave Communications NXT6000 demodulator)
- sp887x : Microtune 7202D
- dib3000mb : DiBcom 3000-MB demodulator

DVB-S/C/T:

- dst : TwinHan DST Frontend

ATSC:

- nxt200x : Nxtwave NXT2002 & NXT2004
- or51211 : or51211 based (pcHDTV HD2000 card)
- or51132 : or51132 based (pcHDTV HD3000 card)
- bcm3510 : Broadcom BCM3510
- lgdt330x : LG Electronics DT3302 & DT3303

- Cards based on the Phillips saa7146 multimedia PCI bridge chip:
  - TI AV7110 based cards (i.e. with hardware MPEG decoder): - Siemens/Technotrend/Hauppauge PCI DVB card revision 1.1, 1.3, 1.5, 1.6, 2.1 (aka Hauppauge Nexus)
  - “budget” cards (i.e. without hardware MPEG decoder): - Technotrend Budget / Hauppauge WinTV-Nova PCI Cards - SATELCO Multimedia PCI - KNC1 DVB-S, Typhoon DVB-S, Terratec Cinergy 1200 DVB-S (no CI support) - Typhoon DVB-S budget - Fujitsu-Siemens Activy DVB-S budget card
- Cards based on the B2C2 Inc. FlexCopII/IIb/III:
  - Technisat SkyStar2 PCI DVB card revision 2.3, 2.6B, 2.6C
- Cards based on the Conexant Bt8xx PCI bridge:
  - Pinnacle PCTV Sat DVB
  - Nebula Electronics DigiTV

  - TwinHan DST
  - Avermedia DVB-T
  - ChainTech digitop DST-1000 DVB-S
  - pcHDTV HD-2000 TV
  - DViCO FusionHDTV DVB-T Lite
  - DViCO FusionHDTV5 Lite
- Technotrend / Hauppauge DVB USB devices:
  - Nova USB
  - DEC 2000-T, 3000-S, 2540-T
- DiBcom DVB-T USB based devices:
  - Twinhan VisionPlus VisionDTV USB-Ter DVB-T Device
  - HAMA DVB-T USB device
  - CTS Portable (Chinese Television System)
  - KWorld V-Stream XPERT DTV DVB-T USB
  - JetWay DTV DVB-T USB
  - ADSTech Instant TV DVB-T USB
  - Ultima Electronic/Artec T1 USB TVBOX (AN2135 and AN2235)
  - Compro Videomate DVB-U2000 - DVB-T USB
  - Grandtec USB DVB-T
  - Avermedia AverTV DVBT USB
  - DiBcom USB DVB-T reference device (non-public)
  - Yakumo DVB-T mobile USB2.0
  - DiBcom USB2.0 DVB-T reference device (non-public)
- Experimental support for the analog module of the Siemens DVB-C PCI card
- Cards based on the Conexant cx2388x PCI bridge:
  - ADS Tech Instant TV DVB-T PCI
  - ATI HDTV Wonder
  - digitalnow DNTV Live! DVB-T
  - DViCO FusionHDTV DVB-T1
  - DViCO FusionHDTV DVB-T Plus
  - DViCO FusionHDTV3 Gold-Q
  - DViCO FusionHDTV3 Gold-T
  - DViCO FusionHDTV5 Gold
  - Hauppauge Nova-T DVB-T
  - KWorld/VStream XPert DVB-T
  - pcHDTV HD3000 HDTV
  - TerraTec Cinergy 1400 DVB-T
  - WinFast DTV1000-T
- Cards based on the Phillips saa7134 PCI bridge:

- Medion 7134
- Pinnacle PCTV 300i DVB-T + PAL
- LifeView FlyDVB-T DUO
- Typhoon DVB-T Duo Digital/Analog Cardbus
- Philips TOUGH DVB-T reference design
- Philips EUROPA V3 reference design
- Compro Videomate DVB-T300
- Compro Videomate DVB-T200
- AVerMedia AVerTVHD MCE A180
- KWorld PC150-U ATSC Hybrid

3.5 Digital TV Conditional Access Interface (CI API)

Note:

This documentation is outdated.

This document describes the usage of the high level CI API as in accordance to the Linux DVB API. This is a not a documentation for the, existing low level CI API.

Note:

For the Twinhan/Twinhan clones, the dst_ca module handles the CI hardware handling.This module is loaded automatically if a CI (Common Interface, that holds the CAM (Conditional Access Module) is detected.

3.5.1 ca_zap

An userspace application, like ca_zap is required to handle encrypted MPEG-TS streams.

The ca_zap userland application is in charge of sending the descrambling related information to the Conditional Access Module (CAM).

This application requires the following to function properly as of now.

1. Tune to a valid channel, with szap.

   eg: $ szap -c channels.conf -r “TMC” -x

2. a channels.conf containing a valid PMT PID

   eg: TMC:11996:h:0:27500:278:512:650:321

   here 278 is a valid PMT PID. the rest of the values are the same ones that szap uses.

3. after running a szap, you have to run ca_zap, for the descrambler to function,

   eg: $ ca_zap channels.conf “TMC”

4. Hopefully enjoy your favourite subscribed channel as you do with a FTA card.

Note:

Currently ca_zap, and dst_test, both are meant for demonstration purposes only, they can become full fledged applications if necessary.

3.5.2 Cards that fall in this category

At present the cards that fall in this category are the Twinhan and its clones, these cards are available as VVMER, Tomato, Hercules, Orange and so on.

3.5.3 CI modules that are supported

The CI module support is largely dependent upon the firmware on the cards Some cards do support almost all of the available CI modules. There is nothing much that can be done in order to make additional CI modules working with these cards.

Modules that have been tested by this driver at present are

1. Irdeto 1 and 2 from SCM
2. Viaccess from SCM
3. Dragoncam

3.5.4 The High level CI API

For the programmer

With the High Level CI approach any new card with almost any random architecture can be implemented with this style, the definitions inside the switch statement can be easily adapted for any card, thereby eliminating the need for any additional ioctls.

The disadvantage is that the driver/hardware has to manage the rest. For the application programmer it would be as simple as sending/receiving an array to/from the CI ioctls as defined in the Linux DVB API. No changes have been made in the API to accommodate this feature.

3.5.5 Why the need for another CI interface?

This is one of the most commonly asked question. Well a nice question. Strictly speaking this is not a new interface.

The CI interface is defined in the DVB API in ca.h as:

```
typedef struct ca_slot_info {
        int num;                /* slot number */

        int type;               /* CA interface this slot supports */
#define CA_CI            1      /* CI high level interface */
#define CA_CI_LINK       2      /* CI link layer level interface */
#define CA_CI_PHYS       4      /* CI physical layer level interface */
#define CA_DESCR         8      /* built-in descrambler */
#define CA_SC          128      /* simple smart card interface */

        unsigned int flags;
#define CA_CI_MODULE_PRESENT 1 /* module (or card) inserted */
#define CA_CI_MODULE_READY   2
} ca_slot_info_t;
```

This CI interface follows the CI high level interface, which is not implemented by most applications. Hence this area is revisited.

This CI interface is quite different in the case that it tries to accommodate all other CI based devices, that fall into the other categories.

This means that this CI interface handles the EN50221 style tags in the Application layer only and no session management is taken care of by the application. The driver/hardware will take care of all that.

This interface is purely an EN50221 interface exchanging APDU's. This means that no session management, link layer or a transport layer do exist in this case in the application to driver communication. It is as simple as that. The driver/hardware has to take care of that.

With this High Level CI interface, the interface can be defined with the regular ioctls.

All these ioctls are also valid for the High level CI interface

#define CA_RESET _IO('o', 128) #define CA_GET_CAP _IOR('o', 129, ca_caps_t) #define CA_GET_SLOT_INFO _IOR('o', 130, ca_slot_info_t) #define CA_GET_DESCR_INFO _IOR('o', 131, ca_descr_info_t) #define CA_GET_MSG _IOR('o', 132, ca_msg_t) #define CA_SEND_MSG _IOW('o', 133, ca_msg_t) #define CA_SET_DESCR _IOW('o', 134, ca_descr_t) #define CA_SET_PID _IOW('o', 135, ca_pid_t)

On querying the device, the device yields information thus:

```
CA_GET_SLOT_INFO
---------------------------
Command = [info]
APP: Number=[1]
APP: Type=[1]
APP: flags=[1]
APP: CI High level interface
APP: CA/CI Module Present

CA_GET_CAP
---------------------------
Command = [caps]
APP: Slots=[1]
APP: Type=[1]
APP: Descrambler keys=[16]
APP: Type=[1]

CA_SEND_MSG
---------------------------
Descriptors(Program Level)=[ 09 06 06 04 05 50 ff f1]
Found CA descriptor @ program level

(20) ES type=[2] ES pid=[201]  ES length =[0 (0x0)]
(25) ES type=[4] ES pid=[301]  ES length =[0 (0x0)]
ca_message length is 25 (0x19) bytes
EN50221 CA MSG=[ 9f 80 32 19 03 01 2d d1 f0 08 01 09 06 06 04 05 50 ff f1 02 e0 c9 00 00 04 e1␣
 ↪2d 00 00]
```

Not all ioctl's are implemented in the driver from the API, the other features of the hardware that cannot be implemented by the API are achieved using the CA_GET_MSG and CA_SEND_MSG ioctls. An EN50221 style wrapper is used to exchange the data to maintain compatibility with other hardware.

```
/* a message to/from a CI-CAM */
typedef struct ca_msg {
        unsigned int index;
        unsigned int type;
        unsigned int length;
        unsigned char msg[256];
} ca_msg_t;
```

The flow of data can be described thus,

```
     App (User)
     -----
     parse
       |
       |
       v
     en50221 APDU (package)
-------------------------------------
|      |                            | High Level CI driver
|      |                            |
|      v                            |
|    en50221 APDU (unpackage)       |
|      |                            |
|      |                            |
|      v                            |
|    sanity checks                  |
|      |                            |
|      |                            |
|      v                            |
|    do (H/W dep)                   |
-------------------------------------
       |    Hardware
       |
       v
```

The High Level CI interface uses the EN50221 DVB standard, following a standard ensures futureproofness.

3.6 Idea behind the dvb-usb-framework

Note:

1. *This documentation is outdated. Please check at the DVB wiki at https://linuxtv.org/wiki for more updated info.*
2. ***deprecated:*** *Newer DVB USB drivers should use the dvb-usb-v2 framework.*

In March 2005 I got the new Twinhan USB2.0 DVB-T device. They provided specs and a firmware.

Quite keen I wanted to put the driver (with some quirks of course) into dibusb. After reading some specs and doing some USB snooping, it realized, that the dibusb-driver would be a complete mess afterwards. So I decided to do it in a different way: With the help of a dvb-usb-framework.

The framework provides generic functions (mostly kernel API calls), such as:

- Transport Stream URB handling in conjunction with dvb-demux-feed-control (bulk and isoc are supported)
- registering the device for the DVB-API
- registering an I2C-adapter if applicable
- remote-control/input-device handling
- firmware requesting and loading (currently just for the Cypress USB controllers)
- other functions/methods which can be shared by several drivers (such as functions for bulk-control-commands)
- TODO: a I2C-chunker. It creates device-specific chunks of register-accesses depending on length of a register and the number of values that can be multi-written and multi-read.

The source code of the particular DVB USB devices does just the communication with the device via the bus. The connection between the DVB-API-functionality is done via callbacks, assigned in a static device-description (struct dvb_usb_device) each device-driver has to have.

For an example have a look in drivers/media/usb/dvb-usb/vp7045*.

Objective is to migrate all the usb-devices (dibusb, cinergyT2, maybe the ttusb; flexcop-usb already benefits from the generic flexcop-device) to use the dvb-usb-lib.

TODO: dynamic enabling and disabling of the pid-filter in regard to number of feeds requested.

3.6.1 Supported devices

See the LinuxTV DVB Wiki at https://linuxtv.org for a complete list of cards/drivers/firmwares: https://linuxtv.org/wiki/index.php/DVB_USB

0. History & News:

   2005-06-30

   - added support for WideView WT-220U (Thanks to Steve Chang)

   2005-05-30

   - added basic isochronous support to the dvb-usb-framework
   - **added support for Conexant Hybrid reference design and Nebula** DigiTV USB

   2005-04-17

   - all dibusb devices ported to make use of the dvb-usb-framework

   2005-04-02

   - re-enabled and improved remote control code.

   2005-03-31

   - ported the Yakumo/Hama/Typhoon DVB-T USB2.0 device to dvb-usb.

   2005-03-30

   - first commit of the dvb-usb-module based on the dibusb-source. First device is a new driver for the TwinhanDTV Alpha / MagicBox II USB2.0-only DVB-T device.
   - (change from dvb-dibusb to dvb-usb)

   2005-03-28

   - added support for the AVerMedia AverTV DVB-T USB2.0 device (Thanks to Glen Harris and Jiun-Kuei Jung, AVerMedia)

   2005-03-14

   - added support for the Typhoon/Yakumo/HAMA DVB-T mobile USB2.0

   2005-02-11

   - added support for the KWorld/ADSTech Instant DVB-T USB2.0. Thanks a lot to Joachim von Caron

   2005-02-02 - added support for the Hauppauge Win-TV Nova-T USB2

   2005-01-31 - distorted streaming is gone for USB1.1 devices

   2005-01-13

   - moved the mirrored pid_filter_table back to dvb-dibusb first almost working version for HanfTek UMT-010 found out, that Yakumo/HAMA/Typhoon are predecessors of the HanfTek UMT-010

   2005-01-10

- refactoring completed, now everything is very delightful
- tuner quirks for some weird devices (Artec T1 AN2235 device has sometimes a Panasonic Tuner assembled). Tunerprobing implemented. Thanks a lot to Gunnar Wittich.

2004-12-29

- after several days of struggling around bug of no returning URBs fixed.

2004-12-26

- refactored the dibusb-driver, splitted into separate files
- i2c-probing enabled

2004-12-06

- possibility for demod i2c-address probing
- new usb IDs (Compro, Artec)

2004-11-23

- merged changes from DiB3000MC_ver2.1
- revised the debugging
- possibility to deliver the complete TS for USB2.0

2004-11-21

- first working version of the dib3000mc/p frontend driver.

2004-11-12

- added additional remote control keys. Thanks to Uwe Hanke.

2004-11-07

- added remote control support. Thanks to David Matthews.

2004-11-05

- added support for a new devices (Grandtec/Avermedia/Artec)
- merged my changes (for dib3000mb/dibusb) to the FE_REFACTORING, because it became HEAD
- moved transfer control (pid filter, fifo control) from usb driver to frontend, it seems better settled there (added xfer_ops-struct)
- created a common files for frontends (mc/p/mb)

2004-09-28

- added support for a new device (Unknown, vendor ID is Hyper-Paltek)

2004-09-20

- added support for a new device (Compro DVB-U2000), thanks to Amaury Demol for reporting
- changed usb TS transfer method (several urbs, stopping transfer before setting a new pid)

2004-09-13

- added support for a new device (Artec T1 USB TVBOX), thanks to Christian Motschke for reporting

2004-09-05

- released the dibusb device and dib3000mb-frontend driver (old news for vp7041.c)

2004-07-15

- found out, by accident, that the device has a TUA6010XS for PLL

2004-07-12

- figured out, that the driver should also work with the CTS Portable (Chinese Television System)

2004-07-08

- firmware-extraction-2.422-problem solved, driver is now working properly with firmware extracted from 2.422
- #if for 2.6.4 (dvb), compile issue
- changed firmware handling, see vp7041.txt sec 1.1

2004-07-02

- some tuner modifications, v0.1, cleanups, first public

2004-06-28

- now using the dvb_dmx_swfilter_packets, everything runs fine now

2004-06-27

- able to watch and switching channels (pre-alpha)
- no section filtering yet

2004-06-06

- first TS received, but kernel oops :/

2004-05-14

- firmware loader is working

2004-05-11

- start writing the driver

3.6.2 How to use?

Firmware

Most of the USB drivers need to download a firmware to the device before start working.

Have a look at the Wikipage for the DVB-USB-drivers to find out, which firmware you need for your device:

https://linuxtv.org/wiki/index.php/DVB_USB

Compiling

Since the driver is in the linux kernel, activating the driver in your favorite config-environment should sufficient. I recommend to compile the driver as module. Hotplug does the rest.

If you use dvb-kernel enter the build-2.6 directory run ‘make’ and ‘insmod.sh load’ afterwards.

Loading the drivers

Hotplug is able to load the driver, when it is needed (because you plugged in the device).

If you want to enable debug output, you have to load the driver manually and from within the dvb-kernel cvs repository.

first have a look, which debug level are available:

```
# modinfo dvb-usb
# modinfo dvb-usb-vp7045

etc.
```

```
modprobe dvb-usb debug=<level>
modprobe dvb-usb-vp7045 debug=<level>
etc.
```

should do the trick.

When the driver is loaded successfully, the firmware file was in the right place and the device is connected, the “Power”-LED should be turned on.

At this point you should be able to start a dvb-capable application. I’m use (t|s)zap, mplayer and dvbscan to test the basics. VDR-xine provides the long-term test scenario.

3.6.3 Known problems and bugs

- Don’t remove the USB device while running an DVB application, your system will go crazy or die most likely.

Adding support for devices

TODO

USB1.1 Bandwidth limitation

A lot of the currently supported devices are USB1.1 and thus they have a maximum bandwidth of about 5-6 MBit/s when connected to a USB2.0 hub. This is not enough for receiving the complete transport stream of a DVB-T channel (which is about 16 MBit/s). Normally this is not a problem, if you only want to watch TV (this does not apply for HDTV), but watching a channel while recording another channel on the same frequency simply does not work very well. This applies to all USB1.1 DVB-T devices, not just the dvb-usb-devices)

The bug, where the TS is distorted by a heavy usage of the device is gone definitely. All dvb-usb-devices I was using (Twinhan, Kworld, DiBcom) are working like charm now with VDR. Sometimes I even was able to record a channel and watch another one.

Comments

Patches, comments and suggestions are very very welcome.

3.6.4 3. Acknowledgements

Amaury Demol (Amaury.Demol@parrot.com) and Francois Kanounnikoff from DiBcom for providing specs, code and help, on which the dvb-dibusb, dib3000mb and dib3000mc are based.

David Matthews for identifying a new device type (Artec T1 with AN2235) and for extending dibusb with remote control event handling. Thank you.

Alex Woods for frequently answering question about usb and dvb stuff, a big thank you.

Bernd Wagner for helping with huge bug reports and discussions.

Gunnar Wittich and Joachim von Caron for their trust for providing root-shells on their machines to implement support for new devices.

Allan Third and Michael Hutchinson for their help to write the Nebula digitv-driver.

Glen Harris for bringing up, that there is a new dibusb-device and Jiun-Kuei Jung from AVerMedia who kindly provided a special firmware to get the device up and running in Linux.

Jennifer Chen, Jeff and Jack from Twinhan for kindly supporting by writing the vp7045-driver.

Steve Chang from WideView for providing information for new devices and firmware files.

Michael Paxton for submitting remote control keymaps.

Some guys on the linux-dvb mailing list for encouraging me.

Peter Schildmann >peter.schildmann-nospam-at-web.de< for his user-level firmware loader, which saves a lot of time (when writing the vp7041 driver)

Ulf Hermenau for helping me out with traditional chinese.

André Smoktun and Christian Frömmel for supporting me with hardware and listening to my problems very patiently.

3.7 FAQ

Note:

This documentation is outdated. Please check at the DVB wiki at https://linuxtv.org/wiki for more updated info.

Some very frequently asked questions about linuxtv-dvb

1. The signal seems to die a few seconds after tuning.

   It's not a bug, it's a feature. Because the frontends have significant power requirements (and hence get very hot), they are powered down if they are unused (i.e. if the frontend device is closed). The dvb-core.o module parameter "dvb_shutdown_timeout" allow you to change the timeout (default 5 seconds). Setting the timeout to 0 disables the timeout feature.

2. How can I watch TV?

   The driver distribution includes some simple utilities which are mainly intended for testing and to demonstrate how the DVB API works.

   Depending on whether you have a DVB-S, DVB-C or DVB-T card, use apps/szap/szap, czap or tzap. You must supply a channel list in ~/.[sct]zap/channels.conf. If you are lucky you can just copy one of the supplied channel lists, or you can create a new one by running apps/scan/scan. If you run scan on an unknown network you might have to supply some start data in apps/scan/initial.h.

   If you have a card with a built-in hardware MPEG-decoder the drivers create a video4linux device (/dev/v4l/video0) which you can use to watch TV with any v4l application. xawtv is known to work. Note that you cannot change channels with xawtv, you have to zap using [sct]zap. If you want a nice application for TV watching and record/playback, have a look at VDR.

   If your card does not have a hardware MPEG decoder you need a software MPEG decoder. Mplayer or xine are known to work. Newsflash: MythTV also has DVB support now. Note: Only very recent versions of Mplayer and xine can decode. MPEG2 transport streams (TS) directly. Then, run '[sct]zap channelname -r' in one xterm, and keep it running, and start 'mplayer - < /dev/dvb/adapter0/dvr0' or 'xine stdin://mpeg2 < /dev/dvb/adapter0/dvr0' in a second xterm. That's all far from perfect, but it seems no one has written a nice DVB application which includes a builtin software MPEG decoder yet.

Newsflash: Newest xine directly supports DVB. Just copy your channels.conf to ~/.xine and start 'xine dvb:///', or select the DVB button in the xine GUI. Channel switching works using the numpad pgup/pgdown (NP9 / NP3) keys to scroll through the channel osd menu and pressing numpad-enter to switch to the selected channel.

Note: Older versions of xine and mplayer understand MPEG program streams (PS) only, and can be used in conjunction with the ts2ps tool from the Metzler Brother's dvb-mpegtools package.

3. Which other DVB applications exist?

   http://www.cadsoft.de/people/kls/vdr/ Klaus Schmidinger's Video Disk Recorder

   http://www.metzlerbros.org/dvb/ Metzler Bros. DVB development; alternate drivers and DVB utilities, include dvb-mpegtools and tuxzap.

   http://sourceforge.net/projects/dvbtools/ Dave Chapman's dvbtools package, including dvbstream and dvbtune

   http://www.linuxdvb.tv/ Henning Holtschneider's site with many interesting links and docs

   http://www.dbox2.info/ LinuxDVB on the dBox2

   http://www.tuxbox.org/ and http://cvs.tuxbox.org/ the TuxBox CVS many interesting DVB applications and the dBox2 DVB source

   https://linuxtv.org/downloads DVB Swiss Army Knife library and utilities

   http://www.nenie.org/misc/mpsys/ MPSYS: a MPEG2 system library and tools

   http://mplayerhq.hu/ mplayer

   http://xine.sourceforge.net/ and http://xinehq.de/ xine

   http://www.mythtv.org/ MythTV - analog TV PVR, but now with DVB support, too (with software MPEG decode)

   http://dvbsnoop.sourceforge.net/ DVB sniffer program to monitor, analyze, debug, dump or view dvb/mpeg/dsm-cc/mhp stream information (TS, PES, SECTION)

4. Can't get a signal tuned correctly

   If you are using a Technotrend/Hauppauge DVB-C card *without* analog module, you might have to use module parameter adac=-1 (dvb-ttpci.o).

5. The dvb_net device doesn't give me any packets at all

   Run tcpdump on the dvb0_0 interface. This sets the interface into promiscuous mode so it accepts any packets from the PID you have configured with the dvbnet utility. Check if there are any packets with the IP addr and MAC addr you have configured with ifconfig.

   If tcpdump doesn't give you any output, check the statistics which ifconfig outputs. (Note: If the MAC address is wrong, dvb_net won't get any input; thus you have to run tcpdump before checking the statistics.) If there are no packets at all then maybe the PID is wrong. If there are error packets, then either the PID is wrong or the stream does not conform to the MPE standard (EN 301 192, http://www.etsi.org/). You can use e.g. dvbsnoop for debugging.

6. The dvb_net device doesn't give me any multicast packets

   Check your routes if they include the multicast address range. Additionally make sure that "source validation by reversed path lookup" is disabled:

```
$ "echo 0 > /proc/sys/net/ipv4/conf/dvb0/rp_filter"
```

7. What the hell are all those modules that need to be loaded?

   For a dvb-ttpci av7110 based full-featured card the following modules are loaded:

- videodev: Video4Linux core module. This is the base module that gives you access to the “analog” tv picture of the av7110 mpeg2 decoder.
- v4l2-common: common functions for Video4Linux-2 drivers
- v4l1-compat: backward compatibility layer for Video4Linux-1 legacy applications
- dvb-core: DVB core module. This provides you with the /dev/dvb/adapter entries
- saa7146: SAA7146 core driver. This is need to access any SAA7146 based card in your system.
- saa7146_vv: SAA7146 video and vbi functions. These are only needed for full-featured cards.
- videobuf-dma-sg: capture helper module for the saa7146_vv driver. This one is responsible to handle capture buffers.
- dvb-ttpci: The main driver for AV7110 based, full-featured DVB-S/C/T cards

3.8 Firmware files for lmedm04 cards

To extract firmware for the DM04/QQBOX you need to copy the following file(s) to this directory.

3.8.1 For DM04+/QQBOX LME2510C (Sharp 7395 Tuner)

The Sharp 7395 driver can be found in windows/system32/drivers

US2A0D.sys (dated 17 Mar 2009)

and run:

```
scripts/get_dvb_firmware lme2510c_s7395
```

will produce dvb-usb-lme2510c-s7395.fw

An alternative but older firmware can be found on the driver disk DVB-S_EN_3.5A in BDADriver/driver

LMEBDA_DVBS7395C.sys (dated 18 Jan 2008)

and run:

```
./get_dvb_firmware lme2510c_s7395_old
```

will produce dvb-usb-lme2510c-s7395.fw

The LG firmware can be found on the driver disk DM04+_5.1A[LG] in BDADriver/driver

3.8.2 For DM04 LME2510 (LG Tuner)

LMEBDA_DVBS.sys (dated 13 Nov 2007)

and run:

```
./get_dvb_firmware lme2510_lg
```

will produce dvb-usb-lme2510-lg.fw

Other LG firmware can be extracted manually from US280D.sys only found in windows/system32/drivers

dd if=US280D.sys ibs=1 skip=42360 count=3924 of=dvb-usb-lme2510-lg.fw

3.8.3 For DM04 LME2510C (LG Tuner)

```
dd if=US280D.sys ibs=1 skip=35200 count=3850 of=dvb-usb-lme2510c-lg.fw
```

The Sharp 0194 tuner driver can be found in windows/system32/drivers

US290D.sys (dated 09 Apr 2009)

3.8.4 For LME2510

```
dd if=US290D.sys ibs=1 skip=36856 count=3976 of=dvb-usb-lme2510-s0194.fw
```

3.8.5 For LME2510C

```
dd if=US290D.sys ibs=1 skip=33152 count=3697 of=dvb-usb-lme2510c-s0194.fw
```

The m88rs2000 tuner driver can be found in windows/system32/drivers

US2B0D.sys (dated 29 Jun 2010)

```
dd if=US2B0D.sys ibs=1 skip=34432 count=3871 of=dvb-usb-lme2510c-rs2000.fw
```

We need to modify id of rs2000 firmware or it will warm boot id 3344:1120.

```
echo -ne \\xF0\\x22 | dd conv=notrunc bs=1 count=2 seek=266 of=dvb-usb-lme2510c-rs2000.fw
```

Copy the firmware file(s) to /lib/firmware

3.9 Opera firmware

Author: Marco Gittler <g.marco@freenet.de>

To extract the firmware for the Opera DVB-S1 USB-Box you need to copy the files:

2830SCap2.sys 2830SLoad2.sys

from the windriver disk into this directory.

Then run:

```
scripts/get_dvb_firmware opera1
```

and after that you have 2 files:

dvb-usb-opera-01.fw dvb-usb-opera1-fpga-01.fw

in here.

Copy them into /lib/firmware/ .

After that the driver can load the firmware (if you have enabled firmware loading in kernel config and have hotplug running).

3.10 How to set up the Technisat/B2C2 Flexcop devices

Note:

This documentation is outdated.

Author: Uwe Bugla <uwe.bugla@gmx.de> August 2009

3.10.1 Find out what device you have

Important Notice: The driver does NOT support Technisat USB 2 devices!

First start your linux box with a shipped kernel:

```
lspci -vvv for a PCI device (lsusb -vvv for an USB device) will show you for example:
02:0b.0 Network controller: Techsan Electronics Co Ltd B2C2 FlexCopII DVB chip /
Technisat SkyStar2 DVB card (rev 02)

dmesg | grep frontend may show you for example:
DVB: registering frontend 0 (Conexant CX24123/CX24109)...
```

3.10.2 Kernel compilation:

If the Flexcop / Technisat is the only DVB / TV / Radio device in your box get rid of unnecessary modules and check this one:

`Multimedia support` => `Customise analog and hybrid tuner modules to build`

In this directory uncheck every driver which is activated there (except `Simple tuner support` for ATSC 3rd generation only -> see case 9 please).

Then please activate:

- Main module part:

  `Multimedia support => DVB/ATSC adapters => Technisat/B2C2 FlexcopII(b) and FlexCopIII adapters`

  1. => `Technisat/B2C2 Air/Sky/Cable2PC PCI` (PCI card) or
  2. => `Technisat/B2C2 Air/Sky/Cable2PC USB` (USB 1.1 adapter) and for troubleshooting purposes:
  3. => `Enable debug for the B2C2 FlexCop drivers`

- Frontend / Tuner / Demodulator module part:

  `Multimedia support => DVB/ATSC adapters` => `Customise the frontend modules to build Customise DVB frontends =>`

  - SkyStar DVB-S Revision 2.3:
    1. => `Zarlink VP310/MT312/ZL10313 based`
    2. => `Generic I2C PLL based tuners`
  - SkyStar DVB-S Revision 2.6:
    1. => `ST STV0299 based`
    2. => `Generic I2C PLL based tuners`
  - SkyStar DVB-S Revision 2.7:

1. => Samsung S5H1420 based
2. => Integrant ITD1000 Zero IF tuner for DVB-S/DSS
3. => ISL6421 SEC controller

- SkyStar DVB-S Revision 2.8:
  1. => Conexant CX24123 based
  2. => Conexant CX24113/CX24128 tuner for DVB-S/DSS
  3. => ISL6421 SEC controller
- AirStar DVB-T card:
  1. => Zarlink MT352 based
  2. => Generic I2C PLL based tuners
- CableStar DVB-C card:
  1. => ST STV0297 based
  2. => Generic I2C PLL based tuners
- AirStar ATSC card 1st generation:
  1. => Broadcom BCM3510
- AirStar ATSC card 2nd generation:
  1. => NxtWave Communications NXT2002/NXT2004 based
  2. => Generic I2C PLL based tuners
- AirStar ATSC card 3rd generation:
  1. => LG Electronics LGDT3302/LGDT3303 based
  2. Multimedia support => Customise analog and hybrid tuner modules to build => Simple tuner support

3.11 TechnoTrend/Hauppauge DEC USB Driver

3.11.1 Driver Status

Supported:

- DEC2000-t
- DEC2450-t
- DEC3000-s
- Video Streaming
- Audio Streaming
- Section Filters
- Channel Zapping
- Hotplug firmware loader

To Do:

- Tuner status information
- DVB network interface
- Streaming video PC->DEC

- Conax support for 2450-t

3.11.2 Getting the Firmware

To download the firmware, use the following commands:

```
scripts/get_dvb_firmware dec2000t
scripts/get_dvb_firmware dec2540t
scripts/get_dvb_firmware dec3000s
```

3.11.3 Hotplug Firmware Loading

Since 2.6 kernels, the firmware is loaded at the point that the driver module is loaded.

Copy the three files downloaded above into the /usr/lib/hotplug/firmware or /lib/firmware directory (depending on configuration of firmware hotplug).

3.12 UDEV rules for DVB

Note:

1. *This documentation is outdated. Udev on modern distributions auto-detect the DVB devices.*
2. ***TODO:*** *change this document to explain how to make DVB devices persistent, as, when a machine has multiple devices, they may be detected on different orders, which could cause apps that relies on the device numbers to fail.*

The DVB subsystem currently registers to the sysfs subsystem using the “class_simple” interface.

This means that only the basic information like module loading parameters are presented through sysfs. Other things that might be interesting are currently **not** available.

Nevertheless it’s now possible to add proper udev rules so that the DVB device nodes are created automatically.

We assume that you have udev already up and running and that have been creating the DVB device nodes manually up to now due to the missing sysfs support.

0. Don’t forget to disable your current method of creating the device nodes manually.

1. Unfortunately, you’ll need a helper script to transform the kernel sysfs device name into the well known dvb adapter / device naming scheme. The script should be called “dvb.sh” and should be placed into a script dir where udev can execute it, most likely /etc/udev/scripts/

So, create a new file /etc/udev/scripts/dvb.sh and add the following:

```
#!/bin/sh
/bin/echo $1 | /bin/sed -e 's,dvb\([0-9]\)\.\([^0-9]*\)\([0-9]\),dvb/adapter\1/\2\3,'
```

Don’t forget to make the script executable with “chmod”.

1. You need to create a proper udev rule that will create the device nodes like you know them. All real distributions out there scan the /etc/udev/rules.d directory for rule files. The main udev configuration file /etc/udev/udev.conf will tell you the directory where the rules are, most likely it’s /etc/udev/rules.d/

Create a new rule file in that directory called “dvb.rule” and add the following line:

```
KERNEL="dvb*", PROGRAM="/etc/udev/scripts/dvb.sh %k", NAME="%c"
```

If you want more control over the device nodes (for example a special group membership) have a look at “man udev”.

For every device that registers to the sysfs subsystem with a “dvb” prefix, the helper script /etc/udev/scripts/dvb.sh is invoked, which will then create the proper device node in your /dev/ directory.

3.13 Contributors

Note:

This documentation is outdated. There are several other DVB contributors that aren’t listed below.

Thanks go to the following people for patches and contributions:

- Michael Hunold <m.hunold@gmx.de>
  - for the initial saa7146 driver and its recent overhaul
- Christian Theiss
  - for his work on the initial Linux DVB driver
- Marcus Metzler <mocm@metzlerbros.de> and Ralph Metzler <rjkm@metzlerbros.de>
  - for their continuing work on the DVB driver
- Michael Holzt <kju@debian.org>
  - for his contributions to the dvb-net driver
- Diego Picciani <d.picciani@novacomp.it>
  - for CyberLogin for Linux which allows logging onto EON (in case you are wondering where CyberLogin is, EON changed its login procedure and CyberLogin is no longer used.)
- Martin Schaller <martin@smurf.franken.de>
  - for patching the cable card decoder driver
- Klaus Schmidinger <Klaus.Schmidinger@cadsoft.de>
  - for various fixes regarding tuning, OSD and CI stuff and his work on VDR
- Steve Brown <sbrown@cortland.com>
  - for his AFC kernel thread
- Christoph Martin <martin@uni-mainz.de>
  - for his LIRC infrared handler
- Andreas Oberritter <obi@linuxtv.org>, Dennis Noermann <dennis.noermann@noernet.de>, Felix Domke <tmbinc@elitedvb.net>, Florian Schirmer <jolt@tuxbox.org>, Ronny Strutz <3des@elitedvb.de>, Wolfram Joost <dbox2@frokaschwei.de> and all the other dbox2 people
  - for many bugfixes in the generic DVB Core, frontend drivers and their work on the dbox2 port of the DVB driver
- Oliver Endriss <o.endriss@gmx.de>
  - for many bugfixes
- Andrew de Quincey <adq_dvb@lidskialf.net>
  - for the tda1004x frontend driver, and various bugfixes
- Peter Schildmann <peter.schildmann@web.de>

  - for the driver for the Technisat SkyStar2 PCI DVB card

- Vadim Catana <skystar@moldova.cc>, Roberto Ragusa <r.ragusa@libero.it> and Augusto Cardoso <augusto@carhil.net>
  - for all the work for the FlexCopII chipset by B2C2,Inc.
- Davor Emard <emard@softhome.net>
  - for his work on the budget drivers, the demux code, the module unloading problems, ...
- Hans-Frieder Vogt <hfvogt@arcor.de>
  - for his work on calculating and checking the crc's for the TechnoTrend/Hauppauge DEC driver firmware
- Michael Dreher <michael@5dot1.de> and Andreas 'randy' Weinberger
  - for the support of the Fujitsu-Siemens Activy budget DVB-S
- Kenneth Aafløy <ke-aa@frisurf.no>
  - for adding support for Typhoon DVB-S budget card
- Ernst Peinlich <e.peinlich@inode.at>
  - for tuning/DiSEqC support for the DEC 3000-s
- Peter Beutner <p.beutner@gmx.net>
  - for the IR code for the ttusb-dec driver
- Wilson Michaels <wilsonmichaels@earthlink.net>
  - for the lgdt330x frontend driver, and various bugfixes
- Michael Krufky <mkrufky@linuxtv.org>
  - for maintaining v4l/dvb inter-tree dependencies
- Taylor Jacob <rtjacob@earthlink.net>
  - for the nxt2002 frontend driver
- Jean-Francois Thibert <jeanfrancois@sagetv.com>
  - for the nxt2004 frontend driver
- Kirk Lapray <kirk.lapray@gmail.com>
  - for the or51211 and or51132 frontend drivers, and for merging the nxt2002 and nxt2004 modules into a single nxt200x frontend driver.

(If you think you should be in this list, but you are not, drop a line to the DVB mailing list)

CHAPTER

FOUR

VIDEO4LINUX (V4L) DRIVER-SPECIFIC DOCUMENTATION

Table of Contents

4.1 Guidelines for Video4Linux pixel format 4CCs

Guidelines for Video4Linux 4CC codes defined using v4l2_fourcc() are specified in this document. First of the characters defines the nature of the pixel format, compression and colour space. The interpretation of the other three characters depends on the first one.

Existing 4CCs may not obey these guidelines.

4.1.1 Raw bayer

The following first characters are used by raw bayer formats:

- B: raw bayer, uncompressed
- b: raw bayer, DPCM compressed
- a: A-law compressed
- u: u-law compressed

2nd character: pixel order

- B: BGGR
- G: GBRG
- g: GRBG
- R: RGGB

3rd character: uncompressed bits-per-pixel 0–9, A–

4th character: compressed bits-per-pixel 0–9, A–

4.2 Infrared remote control support in video4linux drivers

Authors: Gerd Hoffmann, Mauro Carvalho Chehab

4.2.1 Basics

Most analog and digital TV boards support remote controllers. Several of them have a microprocessor that receives the IR carriers, convert into pulse/space sequences and then to scan codes, returning such codes to userspace ("scancode mode"). Other boards return just the pulse/space sequences ("raw mode").

The support for remote controller in scancode mode is provided by the standard Linux input layer. The support for raw mode is provided via LIRC.

In order to check the support and test it, it is suggested to download the v4l-utils. It provides two tools to handle remote controllers:

- ir-keytable: provides a way to query the remote controller, list the protocols it supports, enable in-kernel support for IR decoder or switch the protocol and to test the reception of scan codes;
- ir-ctl: provide tools to handle remote controllers that support raw mode via LIRC interface.

Usually, the remote controller module is auto-loaded when the TV card is detected. However, for a few devices, you need to manually load the ir-kbd-i2c module.

4.2.2 How it works

The modules register the remote as keyboard within the linux input layer, i.e. you'll see the keys of the remote as normal key strokes (if CONFIG_INPUT_KEYBOARD is enabled).

Using the event devices (CONFIG_INPUT_EVDEV) it is possible for applications to access the remote via /dev/input/event<n> devices. The udev/systemd will automatically create the devices. If you install the v4l-utils, it may also automatically load a different keytable than the default one. Please see v4l-utils ir-keytable.1 man page for details.

The ir-keytable tool is nice for trouble shooting, i.e. to check whenever the input device is really present, which of the devices it is, check whenever pressing keys on the remote actually generates events and the like. You can also use any other input utility that changes the keymaps, like the input kbd utility.

4.3 Using with lircd

The latest versions of the lircd daemon supports reading events from the linux input layer (via event device). It also supports receiving IR codes in lirc mode.

4.4 Using without lircd

Xorg recognizes several IR keycodes that have its numerical value lower than 247. With the advent of Wayland, the input driver got updated too, and should now accept all keycodes. Yet, you may want to just reasign the keycodes to something that your favorite media application likes.

This can be done by setting v4l-utils to load your own keytable in runtime. Please read ir-keytable.1 man page for details.

4.5 Tuner drivers

4.5.1 Simple tuner Programming

There are some flavors of Tuner programming APIs. These differ mainly by the bandswitch byte.

- L= LG_API (VHF_LO=0x01, VHF_HI=0x02, UHF=0x08, radio=0x04)
- P= PHILIPS_API (VHF_LO=0xA0, VHF_HI=0x90, UHF=0x30, radio=0x04)
- T= TEMIC_API (VHF_LO=0x02, VHF_HI=0x04, UHF=0x01)
- A= ALPS_API (VHF_LO=0x14, VHF_HI=0x12, UHF=0x11)
- M= PHILIPS_MK3 (VHF_LO=0x01, VHF_HI=0x02, UHF=0x04, radio=0x19)

4.5.2 Tuner Manufacturers

- SAMSUNG Tuner identification: (e.g. TCPM9091PD27)

```
TCP [ABCJLMNQ] 90[89][125] [DP] [ACD] 27 [ABCD]
[ABCJLMNQ]:
  A= BG+DK
  B= BG
  C= I+DK
  J= NTSC-Japan
  L= Secam LL
  M= BG+I+DK
  N= NTSC
  Q= BG+I+DK+LL
[89]: ?
[125]:
  2: No FM
  5: With FM
[DP]:
  D= NTSC
  P= PAL
[ACD]:
  A= F-connector
  C= Phono connector
  D= Din Jack
[ABCD]:
  3-wire/I2C tuning, 2-band/3-band
```

These Tuners are PHILIPS_API compatible.

Philips Tuner identification: (e.g. FM1216MF)

```
F[IRMQ]12[1345]6{MF|ME|MP}
F[IRMQ]:
 FI12x6: Tuner Series
 FR12x6: Tuner + Radio IF
 FM12x6: Tuner + FM
 FQ12x6: special
 FMR12x6: special
 TD15xx: Digital Tuner ATSC
12[1345]6:
 1216: PAL BG
 1236: NTSC
 1246: PAL I
 1256: Pal DK
{MF|ME|MP}
 MF: BG LL w/ Secam (Multi France)
```

```
 ME: BG DK I LL   (Multi Europe)
 MP: BG DK I      (Multi PAL)
 MR: BG DK M (?)
 MG: BG DKI M (?)
MK2 series PHILIPS_API, most tuners are compatible to this one !
MK3 series introduced in 2002 w/ PHILIPS_MK3_API
```

Temic Tuner identification: (.e.g 4006FH5)

```
 4[01][0136][269]F[HYNR]5
  40x2: Tuner (5V/33V), TEMIC_API.
  40x6: Tuner 5V
  41xx: Tuner compact
  40x9: Tuner+FM compact
 [0136]
  xx0x: PAL BG
  xx1x: Pal DK, Secam LL
  xx3x: NTSC
  xx6x: PAL I
 F[HYNR]5
  FH5: Pal BG
  FY5: others
  FN5: multistandard
  FR5: w/ FM radio
 3X xxxx: order number with specific connector
Note: Only 40x2 series has TEMIC_API, all newer tuners have PHILIPS_API.
```

LG Innotek Tuner:

- TPI8NSR11 : NTSC J/M (TPI8NSR01 w/FM) (P,210/497)
- TPI8PSB11 : PAL B/G (TPI8PSB01 w/FM) (P,170/450)
- TAPC-I701 : PAL I (TAPC-I001 w/FM) (P,170/450)
- TPI8PSB12 : PAL D/K+B/G (TPI8PSB02 w/FM) (P,170/450)
- TAPC-H701P: NTSC_JP (TAPC-H001P w/FM) (L,170/450)
- TAPC-G701P: PAL B/G (TAPC-G001P w/FM) (L,170/450)
- TAPC-W701P: PAL I (TAPC-W001P w/FM) (L,170/450)
- TAPC-Q703P: PAL D/K (TAPC-Q001P w/FM) (L,170/450)
- TAPC-Q704P: PAL D/K+I (L,170/450)
- TAPC-G702P: PAL D/K+B/G (L,170/450)
- TADC-H002F: NTSC (L,175/410?; 2-B, C-W+11, W+12-69)
- TADC-M201D: PAL D/K+B/G+I (L,143/425) (sound control at I2C address 0xc8)
- TADC-T003F: NTSC Taiwan (L,175/410?; 2-B, C-W+11, W+12-69)

Suffix:

- P= Standard phono female socket
- D= IEC female socket
- F= F-connector

Other Tuners:

- TCL2002MB-1 : PAL BG + DK =TUNER_LG_PAL_NEW_TAPC
- TCL2002MB-1F: PAL BG + DK w/FM =PHILIPS_PAL
- TCL2002MI-2 : PAL I = ??

ALPS Tuners:

- Most are LG_API compatible
- TSCH6 has ALPS_API (TSCH5 ?)
- TSBE1 has extra API 05,02,08 Control_byte=0xCB Source:[1]

4.6 Cards List

4.6.1 AU0828 cards list

Card number	Card name	USB IDs
0	Unknown board	
1	Hauppauge HVR950Q	2040:7200, 2040:7210, 2040:7217, 2040:721b, 2040:721e, 2040:721f, 2040:7280, 0fd9:0008, 2040:7260, 2040:7213, 2040:7270
2	Hauppauge HVR850	2040:7240
3	DViCO FusionHDTV USB	0fe9:d620
4	Hauppauge HVR950Q rev xxF8	2040:7201, 2040:7211, 2040:7281
5	Hauppauge Woodbury	05e1:0480, 2040:8200

4.6.2 BTTV cards list

Card number	Card name	PCI IDs
0	* **UNKNOWN/GENERIC** *	
1	MIRO PCTV	
2	Hauppauge (bt848)	
3	STB, Gateway P/N 6000699 (bt848)	
4	Intel Create and Share PCI/ Smart Video Recorder III	
5	Diamond DTV2000	
6	AVerMedia TVPhone	
7	MATRIX-Vision MV-Delta	
8	Lifeview FlyVideo II (Bt848) LR26 / MAXI TV Video PCI2 LR26	
9	IMS/IXmicro TurboTV	
10	Hauppauge (bt878)	0070:13eb, 0070:3900, 2636:10
11	MIRO PCTV pro	
12	ADS Technologies Channel Surfer TV (bt848)	
13	AVerMedia TVCapture 98	1461:0002, 1461:0004, 1461:03
14	Aimslab Video Highway Xtreme (VHX)	
15	Zoltrix TV-Max	a1a0:a0fc
16	Prolink Pixelview PlayTV (bt878)	
17	Leadtek WinView 601	
18	AVEC Intercapture	
19	Lifeview FlyVideo II EZ /FlyKit LR38 Bt848 (capture only)	
20	CEI Raffles Card	

Continued on next page

[1] conexant100029b-PCI-Decoder-ApplicationNote.pdf

Table 4.1 – continued from previous page

Card number	Card name	PCI IDs
21	Lifeview FlyVideo 98/ Lucky Star Image World ConferenceTV LR50	
22	Askey CPH050/ Phoebe Tv Master + FM	14ff:3002
23	Modular Technology MM201/MM202/MM205/MM210/MM215 PCTV, bt878	14c7:0101
24	Askey CPH05X/06X (bt878) [many vendors]	144f:3002,
25	Terratec TerraTV+ Version 1.0 (Bt848)/ Terra TValue Version 1.0/ Vobis TV-Boostar	
26	Hauppauge WinCam newer (bt878)	
27	Lifeview FlyVideo 98/ MAXI TV Video PCI2 LR50	
28	Terratec TerraTV+ Version 1.1 (bt878)	153b:1127,
29	Imagenation PXC200	1295:200a
30	Lifeview FlyVideo 98 LR50	1f7f:1850
31	Formac iProTV, Formac ProTV I (bt848)	
32	Intel Create and Share PCI/ Smart Video Recorder III	
33	Terratec TerraTValue Version Bt878	153b:1117,
34	Leadtek WinFast 2000/ WinFast 2000 XP	107d:6606,
35	Lifeview FlyVideo 98 LR50 / Chronos Video Shuttle II	1851:1850,
36	Lifeview FlyVideo 98FM LR50 / Typhoon TView TV/FM Tuner	1852:1852
37	Prolink PixelView PlayTV pro	
38	Askey CPH06X TView99	144f:3000,
39	Pinnacle PCTV Studio/Rave	11bd:0012,
40	STB TV PCI FM, Gateway P/N 6000704 (bt878), 3Dfx VoodooTV 100	10b4:2636,
41	AVerMedia TVPhone 98	1461:0001,
42	ProVideo PV951	aa0c:146c
43	Little OnAir TV	
44	Sigma TVII-FM	
45	MATRIX-Vision MV-Delta 2	
46	Zoltrix Genie TV/FM	15b0:4000,
47	Terratec TV/Radio+	153b:1123
48	Askey CPH03x/ Dynalink Magic TView	
49	IODATA GV-BCTV3/PCI	10fc:4020
50	Prolink PV-BT878P+4E / PixelView PlayTV PAK / Lenco MXTV-9578 CP	
51	Eagle Wireless Capricorn2 (bt878A)	
52	Pinnacle PCTV Studio Pro	
53	Typhoon TView RDS + FM Stereo / KNC1 TV Station RDS	
54	Lifeview FlyVideo 2000 /FlyVideo A2/ Lifetec LT 9415 TV [LR90]	
55	Askey CPH031/ BESTBUY Easy TV	
56	Lifeview FlyVideo 98FM LR50	a051:41a0
57	GrandTec ‘Grand Video Capture’ (Bt848)	4344:4142
58	Askey CPH060/ Phoebe TV Master Only (No FM)	
59	Askey CPH03x TV Capturer	
60	Modular Technology MM100PCTV	
61	AG Electronics GMV1	15cb:0101
62	Askey CPH061/ BESTBUY Easy TV (bt878)	
63	ATI TV-Wonder	1002:0001
64	ATI TV-Wonder VE	1002:0003
65	Lifeview FlyVideo 2000S LR90	
66	Terratec TValueRadio	153b:1135,
67	IODATA GV-BCTV4/PCI	10fc:4050
68	3Dfx VoodooTV FM (Euro)	10b4:2637
69	Active Imaging AIMMS	
70	Prolink Pixelview PV-BT878P+ (Rev.4C,8E)	
71	Lifeview FlyVideo 98EZ (capture only) LR51	1851:1851
72	Prolink Pixelview PV-BT878P+9B (PlayTV Pro rev.9B FM+NICAM)	1554:4011
73	Sensoray 311/611	6000:0311,

Continued on next page

Table 4.1 – continued from previous page

Card number	Card name	PCI IDs
74	RemoteVision MX (RV605)	
75	Powercolor MTV878/ MTV878R/ MTV878F	
76	Canopus WinDVR PCI (COMPAQ Presario 3524JP, 5112JP)	0e11:0079
77	GrandTec Multi Capture Card (Bt878)	
78	Jetway TV/Capture JW-TV878-FBK, Kworld KW-TV878RF	0a01:17de
79	DSP Design TCVIDEO	
80	Hauppauge WinTV PVR	0070:4500
81	IODATA GV-BCTV5/PCI	10fc:4070,
82	Osprey 100/150 (878)	0070:ff00
83	Osprey 100/150 (848)	
84	Osprey 101 (848)	
85	Osprey 101/151	
86	Osprey 101/151 w/ svid	
87	Osprey 200/201/250/251	
88	Osprey 200/250	0070:ff01
89	Osprey 210/220/230	
90	Osprey 500	0070:ff02
91	Osprey 540	0070:ff04
92	Osprey 2000	0070:ff03
93	IDS Eagle	
94	Pinnacle PCTV Sat	11bd:001c
95	Formac ProTV II (bt878)	
96	MachTV	
97	Euresys Picolo	
98	ProVideo PV150	aa00:1460,
99	AD-TVK503	
100	Hercules Smart TV Stereo	
101	Pace TV & Radio Card	
102	IVC-200	0000:a155,
103	Grand X-Guard / Trust 814PCI	0304:0102
104	Nebula Electronics DigiTV	0071:0101
105	ProVideo PV143	aa00:1430,
106	PHYTEC VD-009-X1 VD-011 MiniDIN (bt878)	
107	PHYTEC VD-009-X1 VD-011 Combi (bt878)	
108	PHYTEC VD-009 MiniDIN (bt878)	
109	PHYTEC VD-009 Combi (bt878)	
110	IVC-100	ff00:a132
111	IVC-120G	ff00:a182,
112	pcHDTV HD-2000 TV	7063:2000
113	Twinhan DST + clones	11bd:0026,
114	Winfast VC100	107d:6607
115	Teppro TEV-560/InterVision IV-560	
116	SIMUS GVC1100	aa6a:82b2
117	NGS NGSTV+	
118	LMLBT4	
119	Tekram M205 PRO	
120	Conceptronic CONTVFMi	
121	Euresys Picolo Tetra	1805:0105,
122	Spirit TV Tuner	
123	AVerMedia AVerTV DVB-T 771	1461:0771
124	AverMedia AverTV DVB-T 761	1461:0761
125	MATRIX Vision Sigma-SQ	
126	MATRIX Vision Sigma-SLC	

Continued on next page

Table 4.1 – continued from previous page

Card number	Card name	PCI IDs
127	APAC Viewcomp 878(AMAX)	
128	DViCO FusionHDTV DVB-T Lite	18ac:db10,
129	V-Gear MyVCD	
130	Super TV Tuner	
131	Tibet Systems 'Progress DVR' CS16	
132	Kodicom 4400R (master)	
133	Kodicom 4400R (slave)	
134	Adlink RTV24	
135	DViCO FusionHDTV 5 Lite	18ac:d500
136	Acorp Y878F	9511:1540
137	Conceptronic CTVFMi v2	036e:109e
138	Prolink Pixelview PV-BT878P+ (Rev.2E)	
139	Prolink PixelView PlayTV MPEG2 PV-M4900	
140	Osprey 440	0070:ff07
141	Asound Skyeye PCTV	
142	Sabrent TV-FM (bttv version)	
143	Hauppauge ImpactVCB (bt878)	0070:13eb
144	MagicTV	
145	SSAI Security Video Interface	4149:5353
146	SSAI Ultrasound Video Interface	414a:5353
147	VoodooTV 200 (USA)	121a:3000
148	DViCO FusionHDTV 2	dbc0:d200
149	Typhoon TV-Tuner PCI (50684)	
150	Geovision GV-600	008a:763c
151	Kozumi KTV-01C	
152	Encore ENL TV-FM-2	1000:1801
153	PHYTEC VD-012 (bt878)	
154	PHYTEC VD-012-X1 (bt878)	
155	PHYTEC VD-012-X2 (bt878)	
156	IVCE-8784	0000:f050,
157	Geovision GV-800(S) (master)	800a:763d
158	Geovision GV-800(S) (slave)	800b:763d,
159	ProVideo PV183	1830:1540,
160	Tongwei Video Technology TD-3116	f200:3116
161	Aposonic W-DVR	0279:0228
162	Adlink MPG24	
163	Bt848 Capture 14MHz	
164	CyberVision CV06 (SV)	
165	Kworld V-Stream Xpert TV PVR878	
166	PCI-8604PW	

4.6.3 cx23885 cards list

Card number	Card name	PCI IDs
0	UNKNOWN/GENERIC	0070:3400
1	Hauppauge WinTV-HVR1800lp	0070:7600
2	Hauppauge WinTV-HVR1800	0070:7800, 0070:7801, 0070:7809
3	Hauppauge WinTV-HVR1250	0070:7911
4	DViCO FusionHDTV5 Express	18ac:d500
5	Hauppauge WinTV-HVR1500Q	0070:7790, 0070:7797
6	Hauppauge WinTV-HVR1500	0070:7710, 0070:7717
7	Hauppauge WinTV-HVR1200	0070:71d1, 0070:71d3

Continued on next page

Table 4.2 – continued from previous page

Card number	Card name	PCI IDs
8	Hauppauge WinTV-HVR1700	0070:8101
9	Hauppauge WinTV-HVR1400	0070:8010
10	DViCO FusionHDTV7 Dual Express	18ac:d618
11	DViCO FusionHDTV DVB-T Dual Express	18ac:db78
12	Leadtek Winfast PxDVR3200 H	107d:6681
13	Compro VideoMate E650F	185b:e800
14	TurboSight TBS 6920	6920:8888
15	TeVii S470	d470:9022
16	DVBWorld DVB-S2 2005	0001:2005
17	NetUP Dual DVB-S2 CI	1b55:2a2c
18	Hauppauge WinTV-HVR1270	0070:2211
19	Hauppauge WinTV-HVR1275	0070:2215, 0070:221d, 0070:22f2
20	Hauppauge WinTV-HVR1255	0070:2251, 0070:22f1
21	Hauppauge WinTV-HVR1210	0070:2291, 0070:2295, 0070:2299, 0070:229d, 0070
22	Mygica X8506 DMB-TH	14f1:8651
23	Magic-Pro ProHDTV Extreme 2	14f1:8657
24	Hauppauge WinTV-HVR1850	0070:8541
25	Compro VideoMate E800	1858:e800
26	Hauppauge WinTV-HVR1290	0070:8551
27	Mygica X8558 PRO DMB-TH	14f1:8578
28	LEADTEK WinFast PxTV1200	107d:6f22
29	GoTView X5 3D Hybrid	5654:2390
30	NetUP Dual DVB-T/C-CI RF	1b55:e2e4
31	Leadtek Winfast PxDVR3200 H XC4000	107d:6f39
32	MPX-885	
33	Mygica X8502/X8507 ISDB-T	14f1:8502
34	TerraTec Cinergy T PCIe Dual	153b:117e
35	TeVii S471	d471:9022
36	Hauppauge WinTV-HVR1255	0070:2259
37	Prof Revolution DVB-S2 8000	8000:3034
38	Hauppauge WinTV-HVR4400/HVR5500	0070:c108, 0070:c138, 0070:c1f8
39	AVerTV Hybrid Express Slim HC81R	1461:d939
40	TurboSight TBS 6981	6981:8888
41	TurboSight TBS 6980	6980:8888
42	Leadtek Winfast PxPVR2200	107d:6f21
43	Hauppauge ImpactVCB-e	0070:7133
44	DViCO FusionHDTV DVB-T Dual Express2	18ac:db98
45	DVBSky T9580	4254:9580
46	DVBSky T980C	4254:980c
47	DVBSky S950C	4254:950c
48	Technotrend TT-budget CT2-4500 CI	13c2:3013
49	DVBSky S950	4254:0950
50	DVBSky S952	4254:0952
51	DVBSky T982	4254:0982
52	Hauppauge WinTV-HVR5525	0070:f038
53	Hauppauge WinTV Starburst	0070:c12a
54	ViewCast 260e	1576:0260
55	ViewCast 460e	1576:0460
56	Hauppauge WinTV-QuadHD-DVB	0070:6a28, 0070:6b28
57	Hauppauge WinTV-QuadHD-ATSC	0070:6a18, 0070:6b18

4.6.4 CX88 cards list

Card number	Card name	PCI IDs
0	UNKNOWN/GENERIC	
1	Hauppauge WinTV 34xxx models	0070:3400, 0070:3401
2	GDI Black Gold	14c7:0106, 14c7:0107
3	PixelView	1554:4811
4	ATI TV Wonder Pro	1002:00f8, 1002:00f9
5	Leadtek Winfast 2000XP Expert	107d:6611, 107d:6613
6	AverTV Studio 303 (M126)	1461:000b
7	MSI TV-@nywhere Master	1462:8606
8	Leadtek Winfast DV2000	107d:6620, 107d:6621
9	Leadtek PVR 2000	107d:663b, 107d:663c, 107d:6632, 107d:6630, 107d:6638
10	IODATA GV-VCP3/PCI	10fc:d003
11	Prolink PlayTV PVR	
12	ASUS PVR-416	1043:4823, 1461:c111
13	MSI TV-@nywhere	
14	KWorld/VStream XPert DVB-T	17de:08a6
15	DViCO FusionHDTV DVB-T1	18ac:db00
16	KWorld LTV883RF	
17	DViCO FusionHDTV 3 Gold-Q	18ac:d810, 18ac:d800
18	Hauppauge Nova-T DVB-T	0070:9002, 0070:9001, 0070:9000
19	Conexant DVB-T reference design	14f1:0187
20	Provideo PV259	1540:2580
21	DViCO FusionHDTV DVB-T Plus	18ac:db10, 18ac:db11
22	pcHDTV HD3000 HDTV	7063:3000
23	digitalnow DNTV Live! DVB-T	17de:a8a6
24	Hauppauge WinTV 28xxx (Roslyn) models	0070:2801
25	Digital-Logic MICROSPACE Entertainment Center (MEC)	14f1:0342
26	IODATA GV/BCTV7E	10fc:d035
27	PixelView PlayTV Ultra Pro (Stereo)	
28	DViCO FusionHDTV 3 Gold-T	18ac:d820
29	ADS Tech Instant TV DVB-T PCI	1421:0334
30	TerraTec Cinergy 1400 DVB-T	153b:1166
31	DViCO FusionHDTV 5 Gold	18ac:d500
32	AverMedia UltraTV Media Center PCI 550	1461:8011
33	Kworld V-Stream Xpert DVD	
34	ATI HDTV Wonder	1002:a101
35	WinFast DTV1000-T	107d:665f
36	AVerTV 303 (M126)	1461:000a
37	Hauppauge Nova-S-Plus DVB-S	0070:9201, 0070:9202
38	Hauppauge Nova-SE2 DVB-S	0070:9200
39	KWorld DVB-S 100	17de:08b2, 1421:0341
40	Hauppauge WinTV-HVR1100 DVB-T/Hybrid	0070:9400, 0070:9402
41	Hauppauge WinTV-HVR1100 DVB-T/Hybrid (Low Profile)	0070:9800, 0070:9802
42	digitalnow DNTV Live! DVB-T Pro	1822:0025, 1822:0019
43	KWorld/VStream XPert DVB-T with cx22702	17de:08a1, 12ab:2300
44	DViCO FusionHDTV DVB-T Dual Digital	18ac:db50, 18ac:db54
45	KWorld HardwareMpegTV XPert	17de:0840, 1421:0305
46	DViCO FusionHDTV DVB-T Hybrid	18ac:db40, 18ac:db44
47	pcHDTV HD5500 HDTV	7063:5500
48	Kworld MCE 200 Deluxe	17de:0841
49	PixelView PlayTV P7000	1554:4813
50	NPG Tech Real TV FM Top 10	14f1:0842
51	WinFast DTV2000 H	107d:665e
52	Geniatech DVB-S	14f1:0084
53	Hauppauge WinTV-HVR3000 TriMode Analog/DVB-S/DVB-T	0070:1404, 0070:1400, 0070:1401

Continued on next page

Table 4.3 – continued from previous page

Card number	Card name	PCI IDs
54	Norwood Micro TV Tuner	
55	Shenzhen Tungsten Ages Tech TE-DTV-250 / Swann OEM	c180:c980
56	Hauppauge WinTV-HVR1300 DVB-T/Hybrid MPEG Encoder	0070:9600, 0070:9601, 0070:9602
57	ADS Tech Instant Video PCI	1421:0390
58	Pinnacle PCTV HD 800i	11bd:0051
59	DViCO FusionHDTV 5 PCI nano	18ac:d530
60	Pinnacle Hybrid PCTV	12ab:1788
61	Leadtek TV2000 XP Global	107d:6f18, 107d:6618, 107d:6619
62	PowerColor RA330	14f1:ea3d
63	Geniatech X8000-MT DVBT	14f1:8852
64	DViCO FusionHDTV DVB-T PRO	18ac:db30
65	DViCO FusionHDTV 7 Gold	18ac:d610
66	Prolink Pixelview MPEG 8000GT	1554:4935
67	Kworld PlusTV HD PCI 120 (ATSC 120)	17de:08c1
68	Hauppauge WinTV-HVR4000 DVB-S/S2/T/Hybrid	0070:6900, 0070:6904, 0070:6902
69	Hauppauge WinTV-HVR4000(Lite) DVB-S/S2	0070:6905, 0070:6906
70	TeVii S460 DVB-S/S2	d460:9022
71	Omicom SS4 DVB-S/S2 PCI	A044:2011
72	TBS 8920 DVB-S/S2	8920:8888
73	TeVii S420 DVB-S	d420:9022
74	Prolink Pixelview Global Extreme	1554:4976
75	PROF 7300 DVB-S/S2	B033:3033
76	SATTRADE ST4200 DVB-S/S2	b200:4200
77	TBS 8910 DVB-S	8910:8888
78	Prof 6200 DVB-S	b022:3022
79	Terratec Cinergy HT PCI MKII	153b:1177
80	Hauppauge WinTV-IR Only	0070:9290
81	Leadtek WinFast DTV1800 Hybrid	107d:6654
82	WinFast DTV2000 H rev. J	107d:6f2b
83	Prof 7301 DVB-S/S2	b034:3034
84	Samsung SMT 7020 DVB-S	18ac:dc00, 18ac:dccd
85	Twinhan VP-1027 DVB-S	1822:0023
86	TeVii S464 DVB-S/S2	d464:9022
87	Leadtek WinFast DTV2000 H PLUS	107d:6f42
88	Leadtek WinFast DTV1800 H (XC4000)	107d:6f38
89	Leadtek TV2000 XP Global (SC4100)	107d:6f36
90	Leadtek TV2000 XP Global (XC4100)	107d:6f43

4.6.5 EM28xx cards list

Card number	Card name	Empia Chip	USB IDs
0	Unknown EM2800 video grabber	em2800	
1	Unknown EM2750/28xx video grabber	em2820 or em2840	
2	Terratec Cinergy 250 USB	em2820 or em2840	
3	Pinnacle PCTV USB 2	em2820 or em2840	
4	Hauppauge WinTV USB 2	em2820 or em2840	
5	MSI VOX USB 2.0	em2820 or em2840	
6	Terratec Cinergy 200 USB	em2800	
7	Leadtek Winfast USB II	em2800	
8	Kworld USB2800	em2800	
9	Pinnacle Dazzle DVC 90/100/101/107 / Kaiser Baas Video to DVD maker	em2820 or em2840	
10	Hauppauge WinTV HVR 900	em2880	

Continued on next page

Table 4.4 – continued from previous page

Card number	Card name	Empia Chip	USB IDs
11	Terratec Hybrid XS	em2880	
12	Kworld PVR TV 2800 RF	em2820 or em2840	
13	Terratec Prodigy XS	em2880	
14	SIIG AVTuner-PVR / Pixelview Prolink PlayTV USB 2.0	em2820 or em2840	
15	V-Gear PocketTV	em2800	
16	Hauppauge WinTV HVR 950	em2883	
17	Pinnacle PCTV HD Pro Stick	em2880	
18	Hauppauge WinTV HVR 900 (R2)	em2880	
19	EM2860/SAA711X Reference Design	em2860	
20	AMD ATI TV Wonder HD 600	em2880	
21	eMPIA Technology, Inc. GrabBeeX+ Video Encoder	em2800	
22	EM2710/EM2750/EM2751 webcam grabber	em2750	
23	Huaqi DLCW-130	em2750	
24	D-Link DUB-T210 TV Tuner	em2820 or em2840	
25	Gadmei UTV310	em2820 or em2840	
26	Hercules Smart TV USB 2.0	em2820 or em2840	
27	Pinnacle PCTV USB 2 (Philips FM1216ME)	em2820 or em2840	
28	Leadtek Winfast USB II Deluxe	em2820 or em2840	
29	EM2860/TVP5150 Reference Design	em2860	
30	Videology 20K14XUSB USB2.0	em2820 or em2840	
31	Usbgear VD204v9	em2821	
32	Supercomp USB 2.0 TV	em2821	
33	Elgato Video Capture	em2860	
34	Terratec Cinergy A Hybrid XS	em2860	
35	Typhoon DVD Maker	em2860	
36	NetGMBH Cam	em2860	
37	Gadmei UTV330	em2860	
38	Yakumo MovieMixer	em2861	
39	KWorld PVRTV 300U	em2861	
40	Plextor ConvertX PX-TV100U	em2861	
41	Kworld 350 U DVB-T	em2870	
42	Kworld 355 U DVB-T	em2870	
43	Terratec Cinergy T XS	em2870	
44	Terratec Cinergy T XS (MT2060)	em2870	
45	Pinnacle PCTV DVB-T	em2870	
46	Compro, VideoMate U3	em2870	
47	KWorld DVB-T 305U	em2880	
48	KWorld DVB-T 310U	em2880	
49	MSI DigiVox A/D	em2880	
50	MSI DigiVox A/D II	em2880	
51	Terratec Hybrid XS Secam	em2880	
52	DNT DA2 Hybrid	em2881	
53	Pinnacle Hybrid Pro	em2881	
54	Kworld VS-DVB-T 323UR	em2882	
55	Terratec Cinnergy Hybrid T USB XS (em2882)	em2882	
56	Pinnacle Hybrid Pro (330e)	em2882	
57	Kworld PlusTV HD Hybrid 330	em2883	
58	Compro VideoMate ForYou/Stereo	em2820 or em2840	
59	Pinnacle PCTV HD Mini	em2874	
60	Hauppauge WinTV HVR 850	em2883	
61	Pixelview PlayTV Box 4 USB 2.0	em2820 or em2840	
62	Gadmei TVR200	em2820 or em2840	
63	Kaiomy TVnPC U2	em2860	

Continued on next page

Table 4.4 – continued from previous page

Card number	Card name	Empia Chip	USB IDs
64	Easy Cap Capture DC-60	em2860	
65	IO-DATA GV-MVP/SZ	em2820 or em2840	
66	Empire dual TV	em2880	
67	Terratec Grabby	em2860	
68	Terratec AV350	em2860	
69	KWorld ATSC 315U HDTV TV Box	em2882	
70	Evga inDtube	em2882	
71	Silvercrest Webcam 1.3mpix	em2820 or em2840	
72	Gadmei UTV330+	em2861	
73	Reddo DVB-C USB TV Box	em2870	
74	Actionmaster/LinXcel/Digitus VC211A	em2800	
75	Dikom DK300	em2882	
76	KWorld PlusTV 340U or UB435-Q (ATSC)	em2870	
77	EM2874 Leadership ISDBT	em2874	
78	PCTV nanoStick T2 290e	em28174	
79	Terratec Cinergy H5	em2884	
80	PCTV DVB-S2 Stick (460e)	em28174	
81	Hauppauge WinTV HVR 930C	em2884	
82	Terratec Cinergy HTC Stick	em2884	
83	Honestech Vidbox NW03	em2860	
84	MaxMedia UB425-TC	em2874	
85	PCTV QuatroStick (510e)	em2884	
86	PCTV QuatroStick nano (520e)	em2884	
87	Terratec Cinergy HTC USB XS	em2884	
88	C3 Tech Digital Duo HDTV/SDTV USB	em2884	
89	Delock 61959	em2874	
90	KWorld USB ATSC TV Stick UB435-Q V2	em2874	
91	SpeedLink Vicious And Devine Laplace webcam	em2765	
92	PCTV DVB-S2 Stick (461e)	em28178	
93	KWorld USB ATSC TV Stick UB435-Q V3	em2874	
94	PCTV tripleStick (292e)	em28178	
95	Leadtek VC100	em2861	
96	Terratec Cinergy T2 Stick HD	em28178	
97	Elgato EyeTV Hybrid 2008 INT	em2884	
98	PLEX PX-BCUD	em28178	
99	Hauppauge WinTV-dualHD DVB	em28174	

4.6.6 IVTV cards list

Card number	Card name	PCI IDs
0	Hauppauge WinTV PVR-250	IVTV16 104d:813d
1	Hauppauge WinTV PVR-350	IVTV16 104d:813d
2	Hauppauge WinTV PVR-150	IVTV16 104d:813d
3	AVerMedia M179	IVTV15 1461:a3cf, IVTV1
4	Yuan MPG600, Kuroutoshikou ITVC16-STVLP	IVTV16 12ab:fff3, IVTV16
5	YUAN MPG160, Kuroutoshikou ITVC15-STVLP, I/O Data GV-M2TV/PCI	IVTV15 10fc:40a0
6	Yuan PG600, Diamond PVR-550	IVTV16 ff92:0070, IVTV1
7	Adaptec VideOh! AVC-2410	IVTV16 9005:0093
8	Adaptec VideOh! AVC-2010	IVTV16 9005:0092
9	Nagase Transgear 5000TV	IVTV16 1461:bfff
10	AOpen VA2000MAX-SNT6	IVTV16 0000:ff5f
11	Yuan MPG600GR, Kuroutoshikou CX23416GYC-STVLP	IVTV16 12ab:0600, IVTV1

Continued on next page

Table 4.5 – continued from previous page

Card number	Card name	PCI IDs
12	I/O Data GV-MVP/RX, GV-MVP/RX2W (dual tuner)	IVTV16 10fc:d01e, IVTV1
13	I/O Data GV-MVP/RX2E	IVTV16 10fc:d025
14	GotView PCI DVD	IVTV16 12ab:0600
15	GotView PCI DVD2 Deluxe	IVTV16 ffac:0600
16	Yuan MPC622	IVTV16 ff01:d998
17	Digital Cowboy DCT-MTVP1	IVTV16 1461:bfff
18	Yuan PG600-2, GotView PCI DVD Lite	IVTV16 ffab:0600, IVTV1
19	Club3D ZAP-TV1x01	IVTV16 ffab:0600
20	AVerTV MCE 116 Plus	IVTV16 1461:c439
21	ASUS Falcon2	IVTV16 1043:4b66, IVTV
22	AVerMedia PVR-150 Plus / AVerTV M113 Partsnic (Daewoo) Tuner	IVTV16 1461:c034, IVTV
23	AVerMedia EZMaker PCI Deluxe	IVTV16 1461:c03f
24	AVerMedia M104	IVTV16 1461:c136
25	Buffalo PC-MV5L/PCI	IVTV16 1154:052b
26	AVerMedia UltraTV 1500 MCE / AVerTV M113 Philips Tuner	IVTV16 1461:c019, IVTV
27	Sony VAIO Giga Pocket (ENX Kikyou)	IVTV16 104d:813d
28	Hauppauge WinTV PVR-350 (V1)	IVTV16 104d:813d
29	Yuan MPG600GR, Kuroutoshikou CX23416GYC-STVLP (no GR)	IVTV16 104d:813d
30	Yuan MPG600GR, Kuroutoshikou CX23416GYC-STVLP (no GR/YCS)	IVTV16 104d:813d

4.6.7 SAA7134 cards list

Card number	Card name	PCI IDs
0	UNKNOWN/GENERIC	
1	Proteus Pro [philips reference design]	1131:2001, 1131:2001
2	LifeView FlyVIDEO3000	5168:0138, 4e42:0138
3	LifeView/Typhoon FlyVIDEO2000	5168:0138, 4e42:0138
4	EMPRESS	1131:6752
5	SKNet Monster TV	1131:4e85
6	Tevion MD 9717	
7	KNC One TV-Station RDS / Typhoon TV Tuner RDS	1131:fe01, 1894:fe01
8	Terratec Cinergy 400 TV	153b:1142
9	Medion 5044	
10	Kworld/KuroutoShikou SAA7130-TVPCI	
11	Terratec Cinergy 600 TV	153b:1143
12	Medion 7134	16be:0003, 16be:5000
13	Typhoon TV+Radio 90031	
14	ELSA EX-VISION 300TV	1048:226b
15	ELSA EX-VISION 500TV	1048:226a
16	ASUS TV-FM 7134	1043:4842, 1043:4830, 1043:4840
17	AOPEN VA1000 POWER	1131:7133
18	BMK MPEX No Tuner	
19	Compro VideoMate TV	185b:c100
20	Matrox CronosPlus	102B:48d0
21	10MOONS PCI TV CAPTURE CARD	1131:2001
22	AverMedia M156 / Medion 2819	1461:a70b
23	BMK MPEX Tuner	
24	KNC One TV-Station DVR	1894:a006
25	ASUS TV-FM 7133	1043:4843
26	Pinnacle PCTV Stereo (saa7134)	11bd:002b
27	Manli MuchTV M-TV002	
28	Manli MuchTV M-TV001	

Continued on next page

Table 4.6 – continued from previous page

Card number	Card name	PCI IDs
29	Nagase Sangyo TransGear 3000TV	1461:050c
30	Elitegroup ECS TVP3XP FM1216 Tuner Card(PAL-BG,FM)	1019:4cb4
31	Elitegroup ECS TVP3XP FM1236 Tuner Card (NTSC,FM)	1019:4cb5
32	AVACS SmartTV	
33	AVerMedia DVD EZMaker	1461:10ff
34	Noval Prime TV 7133	
35	AverMedia AverTV Studio 305	1461:2115
36	UPMOST PURPLE TV	12ab:0800
37	Items MuchTV Plus / IT-005	
38	Terratec Cinergy 200 TV	153b:1152
39	LifeView FlyTV Platinum Mini	5168:0212, 4e42:0212, 5169:1502
40	Compro VideoMate TV PVR/FM	185b:c100
41	Compro VideoMate TV Gold+	185b:c100
42	Sabrent SBT-TVFM (saa7130)	
43	:Zolid Xpert TV7134	
44	Empire PCI TV-Radio LE	
45	Avermedia AVerTV Studio 307	1461:9715
46	AVerMedia Cardbus TV/Radio (E500)	1461:d6ee
47	Terratec Cinergy 400 mobile	153b:1162
48	Terratec Cinergy 600 TV MK3	153b:1158
49	Compro VideoMate Gold+ Pal	185b:c200
50	Pinnacle PCTV 300i DVB-T + PAL	11bd:002d
51	ProVideo PV952	1540:9524
52	AverMedia AverTV/305	1461:2108
53	ASUS TV-FM 7135	1043:4845
54	LifeView FlyTV Platinum FM / Gold	5168:0214, 5168:5214, 1489:0214, 5
55	LifeView FlyDVB-T DUO / MSI TV@nywhere Duo	5168:0306, 4E42:0306
56	Avermedia AVerTV 307	1461:a70a
57	Avermedia AVerTV GO 007 FM	1461:f31f
58	ADS Tech Instant TV (saa7135)	1421:0350, 1421:0351, 1421:0370, 1
59	Kworld/Tevion V-Stream Xpert TV PVR7134	
60	LifeView/Typhoon/Genius FlyDVB-T Duo Cardbus	5168:0502, 4e42:0502, 1489:0502
61	Philips TOUGH DVB-T reference design	1131:2004
62	Compro VideoMate TV Gold+II	
63	Kworld Xpert TV PVR7134	
64	FlyTV mini Asus Digimatrix	1043:0210
65	V-Stream Studio TV Terminator	
66	Yuan TUN-900 (saa7135)	
67	Beholder BeholdTV 409 FM	0000:4091
68	GoTView 7135 PCI	5456:7135
69	Philips EUROPA V3 reference design	1131:2004
70	Compro Videomate DVB-T300	185b:c900
71	Compro Videomate DVB-T200	185b:c901
72	RTD Embedded Technologies VFG7350	1435:7350
73	RTD Embedded Technologies VFG7330	1435:7330
74	LifeView FlyTV Platinum Mini2	14c0:1212
75	AVerMedia AVerTVHD MCE A180	1461:1044
76	SKNet MonsterTV Mobile	1131:4ee9
77	Pinnacle PCTV 40i/50i/110i (saa7133)	11bd:002e
78	ASUSTeK P7131 Dual	1043:4862
79	Sedna/MuchTV PC TV Cardbus TV/Radio (ITO25 Rev:2B)	
80	ASUS Digimatrix TV	1043:0210
81	Philips Tiger reference design	1131:2018

Continued on next page

Table 4.6 – continued from previous page

Card number	Card name	PCI IDs
82	MSI TV@Anywhere plus	1462:6231, 1462:8624
83	Terratec Cinergy 250 PCI TV	153b:1160
84	LifeView FlyDVB Trio	5168:0319
85	AverTV DVB-T 777	1461:2c05, 1461:2c05
86	LifeView FlyDVB-T / Genius VideoWonder DVB-T	5168:0301, 1489:0301
87	ADS Instant TV Duo Cardbus PTV331	0331:1421
88	Tevion/KWorld DVB-T 220RF	17de:7201
89	ELSA EX-VISION 700TV	1048:226c
90	Kworld ATSC110/115	17de:7350, 17de:7352
91	AVerMedia A169 B	1461:7360
92	AVerMedia A169 B1	1461:6360
93	Medion 7134 Bridge #2	16be:0005
94	LifeView FlyDVB-T Hybrid Cardbus/MSI TV @nywhere A/D NB	5168:3306, 5168:3502, 5168:33
95	LifeView FlyVIDEO3000 (NTSC)	5169:0138
96	Medion Md8800 Quadro	16be:0007, 16be:0008, 16be:00
97	LifeView FlyDVB-S /Acorp TV134DS	5168:0300, 4e42:0300
98	Proteus Pro 2309	0919:2003
99	AVerMedia TV Hybrid A16AR	1461:2c00
100	Asus Europa2 OEM	1043:4860
101	Pinnacle PCTV 310i	11bd:002f
102	Avermedia AVerTV Studio 507	1461:9715
103	Compro Videomate DVB-T200A	
104	Hauppauge WinTV-HVR1110 DVB-T/Hybrid	0070:6700, 0070:6701, 0070:67
105	Terratec Cinergy HT PCMCIA	153b:1172
106	Encore ENLTV	1131:2342, 1131:2341, 3016:23
107	Encore ENLTV-FM	1131:230f
108	Terratec Cinergy HT PCI	153b:1175
109	Philips Tiger - S Reference design	
110	Avermedia M102	1461:f31e
111	ASUS P7131 4871	1043:4871
112	ASUSTeK P7131 Hybrid	1043:4876
113	Elitegroup ECS TVP3XP FM1246 Tuner Card (PAL,FM)	1019:4cb6
114	KWorld DVB-T 210	17de:7250
115	Sabrent PCMCIA TV-PCB05	0919:2003
116	10MOONS TM300 TV Card	1131:2304
117	Avermedia Super 007	1461:f01d
118	Beholder BeholdTV 401	0000:4016
119	Beholder BeholdTV 403	0000:4036
120	Beholder BeholdTV 403 FM	0000:4037
121	Beholder BeholdTV 405	0000:4050
122	Beholder BeholdTV 405 FM	0000:4051
123	Beholder BeholdTV 407	0000:4070
124	Beholder BeholdTV 407 FM	0000:4071
125	Beholder BeholdTV 409	0000:4090
126	Beholder BeholdTV 505 FM	5ace:5050
127	Beholder BeholdTV 507 FM / BeholdTV 509 FM	5ace:5070, 5ace:5090
128	Beholder BeholdTV Columbus TV/FM	0000:5201
129	Beholder BeholdTV 607 FM	5ace:6070
130	Beholder BeholdTV M6	5ace:6190
131	Twinhan Hybrid DTV-DVB 3056 PCI	1822:0022
132	Genius TVGO AM11MCE	
133	NXP Snake DVB-S reference design	
134	Medion/Creatix CTX953 Hybrid	16be:0010

Continued on next page

Table 4.6 – continued from previous page

Card number	Card name	PCI IDs
135	MSI TV@nywhere A/D v1.1	1462:8625
136	AVerMedia Cardbus TV/Radio (E506R)	1461:f436
137	AVerMedia Hybrid TV/Radio (A16D)	1461:f936
138	Avermedia M115	1461:a836
139	Compro VideoMate T750	185b:c900
140	Avermedia DVB-S Pro A700	1461:a7a1
141	Avermedia DVB-S Hybrid+FM A700	1461:a7a2
142	Beholder BeholdTV H6	5ace:6290
143	Beholder BeholdTV M63	5ace:6191
144	Beholder BeholdTV M6 Extra	5ace:6193
145	AVerMedia MiniPCI DVB-T Hybrid M103	1461:f636, 1461:f736
146	ASUSTeK P7131 Analog	
147	Asus Tiger 3in1	1043:4878
148	Encore ENLTV-FM v5.3	1a7f:2008
149	Avermedia PCI pure analog (M135A)	1461:f11d
150	Zogis Real Angel 220	
151	ADS Tech Instant HDTV	1421:0380
152	Asus Tiger Rev:1.00	1043:4857
153	Kworld Plus TV Analog Lite PCI	17de:7128
154	Avermedia AVerTV GO 007 FM Plus	1461:f31d
155	Hauppauge WinTV-HVR1150 ATSC/QAM-Hybrid	0070:6706, 0070:6708
156	Hauppauge WinTV-HVR1120 DVB-T/Hybrid	0070:6707, 0070:6709, 0070:67
157	Avermedia AVerTV Studio 507UA	1461:a11b
158	AVerMedia Cardbus TV/Radio (E501R)	1461:b7e9
159	Beholder BeholdTV 505 RDS	0000:505B
160	Beholder BeholdTV 507 RDS	0000:5071
161	Beholder BeholdTV 507 RDS	0000:507B
162	Beholder BeholdTV 607 FM	5ace:6071
163	Beholder BeholdTV 609 FM	5ace:6090
164	Beholder BeholdTV 609 FM	5ace:6091
165	Beholder BeholdTV 607 RDS	5ace:6072
166	Beholder BeholdTV 607 RDS	5ace:6073
167	Beholder BeholdTV 609 RDS	5ace:6092
168	Beholder BeholdTV 609 RDS	5ace:6093
169	Compro VideoMate S350/S300	185b:c900
170	AverMedia AverTV Studio 505	1461:a115
171	Beholder BeholdTV X7	5ace:7595
172	RoverMedia TV Link Pro FM	19d1:0138
173	Zolid Hybrid TV Tuner PCI	1131:2004
174	Asus Europa Hybrid OEM	1043:4847
175	Leadtek Winfast DTV1000S	107d:6655
176	Beholder BeholdTV 505 RDS	0000:5051
177	Hawell HW-404M7	
178	Beholder BeholdTV H7	5ace:7190
179	Beholder BeholdTV A7	5ace:7090
180	Avermedia PCI M733A	1461:4155, 1461:4255
181	TechoTrend TT-budget T-3000	13c2:2804
182	Kworld PCI SBTVD/ISDB-T Full-Seg Hybrid	17de:b136
183	Compro VideoMate Vista M1F	185b:c900
184	Encore ENLTV-FM 3	1a7f:2108
185	MagicPro ProHDTV Pro2 DMB-TH/Hybrid	17de:d136
186	Beholder BeholdTV 501	5ace:5010
187	Beholder BeholdTV 503 FM	5ace:5030

Continued on next page

Table 4.6 – continued from previous page

Card number	Card name	PCI IDs
188	Sensoray 811/911	6000:0811, 6000:0911
189	Kworld PC150-U	17de:a134
190	Asus My Cinema PS3-100	1043:48cd
191	Hawell HW-9004V1	
192	AverMedia AverTV Satellite Hybrid+FM A706	1461:2055
193	WIS Voyager or compatible	1905:7007
194	AverMedia AverTV/505	1461:a10a
195	Leadtek Winfast TV2100 FM	107d:6f3a
196	SnaZio* TVPVR PRO	1779:13cf

4.6.8 SAA7164 cards list

Card number	Card name	PCI IDs
0	Unknown	
1	Generic Rev2	
2	Generic Rev3	
3	Hauppauge WinTV-HVR2250	0070:8880, 0070:8810
4	Hauppauge WinTV-HVR2200	0070:8980
5	Hauppauge WinTV-HVR2200	0070:8900
6	Hauppauge WinTV-HVR2200	0070:8901
7	Hauppauge WinTV-HVR2250	0070:8891, 0070:8851
8	Hauppauge WinTV-HVR2250	0070:88A1
9	Hauppauge WinTV-HVR2200	0070:8940
10	Hauppauge WinTV-HVR2200	0070:8953
11	Hauppauge WinTV-HVR2255(proto)	0070:f111
12	Hauppauge WinTV-HVR2255	0070:f111
13	Hauppauge WinTV-HVR2205	0070:f123, 0070:f120

4.6.9 TM6000 cards list

Card number	Card name	USB IDs
0	Unknown tm6000 video grabber	
1	Generic tm5600 board	6000:0001
2	Generic tm6000 board	
3	Generic tm6010 board	6000:0002
4	10Moons UT 821	
5	10Moons UT 330	
6	ADSTECH Dual TV USB	06e1:f332
7	Freecom Hybrid Stick / Moka DVB-T Receiver Dual	14aa:0620
8	ADSTECH Mini Dual TV USB	06e1:b339
9	Hauppauge WinTV HVR-900H / WinTV USB2-Stick	2040:6600, 2040:6601, 2040:6610, 2040:6611
10	Beholder Wander DVB-T/TV/FM USB2.0	6000:dec0
11	Beholder Voyager TV/FM USB2.0	6000:dec1
12	Terratec Cinergy Hybrid XE / Cinergy Hybrid-Stick	0ccd:0086, 0ccd:00A5
13	Twinhan TU501(704D1)	13d3:3240, 13d3:3241, 13d3:3243, 13d3:3264
14	Beholder Wander Lite DVB-T/TV/FM USB2.0	6000:dec2
15	Beholder Voyager Lite TV/FM USB2.0	6000:dec3
16	Terratec Grabster AV 150/250 MX	0ccd:0079

4.6.10 Tuner cards list

Tuner number	Card name
0	Temic PAL (4002 FH5)
1	Philips PAL_I (FI1246 and compatibles)
2	Philips NTSC (FI1236,FM1236 and compatibles)
3	Philips (SECAM+PAL_BG) (FI1216MF, FM1216MF, FR1216MF)
4	NoTuner
5	Philips PAL_BG (FI1216 and compatibles)
6	Temic NTSC (4032 FY5)
7	Temic PAL_I (4062 FY5)
8	Temic NTSC (4036 FY5)
9	Alps HSBH1
10	Alps TSBE1
11	Alps TSBB5
12	Alps TSBE5
13	Alps TSBC5
14	Temic PAL_BG (4006FH5)
15	Alps TSCH6
16	Temic PAL_DK (4016 FY5)
17	Philips NTSC_M (MK2)
18	Temic PAL_I (4066 FY5)
19	Temic PAL* auto (4006 FN5)
20	Temic PAL_BG (4009 FR5) or PAL_I (4069 FR5)
21	Temic NTSC (4039 FR5)
22	Temic PAL/SECAM multi (4046 FM5)
23	Philips PAL_DK (FI1256 and compatibles)
24	Philips PAL/SECAM multi (FQ1216ME)
25	LG PAL_I+FM (TAPC-I001D)
26	LG PAL_I (TAPC-I701D)
27	LG NTSC+FM (TPI8NSR01F)
28	LG PAL_BG+FM (TPI8PSB01D)
29	LG PAL_BG (TPI8PSB11D)
30	Temic PAL* auto + FM (4009 FN5)
31	SHARP NTSC_JP (2U5JF5540)
32	Samsung PAL TCPM9091PD27
33	MT20xx universal
34	Temic PAL_BG (4106 FH5)
35	Temic PAL_DK/SECAM_L (4012 FY5)
36	Temic NTSC (4136 FY5)
37	LG PAL (newer TAPC series)
38	Philips PAL/SECAM multi (FM1216ME MK3)
39	LG NTSC (newer TAPC series)
40	HITACHI V7-J180AT
41	Philips PAL_MK (FI1216 MK)
42	Philips FCV1236D ATSC/NTSC dual in
43	Philips NTSC MK3 (FM1236MK3 or FM1236/F)
44	Philips 4 in 1 (ATI TV Wonder Pro/Conexant)
45	Microtune 4049 FM5
46	Panasonic VP27s/ENGE4324D
47	LG NTSC (TAPE series)
48	Tenna TNF 8831 BGFF)
49	Microtune 4042 FI5 ATSC/NTSC dual in
50	TCL 2002N
51	Philips PAL/SECAM_D (FM 1256 I-H3)

Continued on next page

Table 4.7 – continued from previous page

Tuner number	Card name
52	Thomson DTT 7610 (ATSC/NTSC)
53	Philips FQ1286
54	Philips/NXP TDA 8290/8295 + 8275/8275A/18271
55	TCL 2002MB
56	Philips PAL/SECAM multi (FQ1216AME MK4)
57	Philips FQ1236A MK4
58	Ymec TVision TVF-8531MF/8831MF/8731MF
59	Ymec TVision TVF-5533MF
60	Thomson DTT 761X (ATSC/NTSC)
61	Tena TNF9533-D/IF/TNF9533-B/DF
62	Philips TEA5767HN FM Radio
63	Philips FMD1216ME MK3 Hybrid Tuner
64	LG TDVS-H06xF
65	Ymec TVF66T5-B/DFF
66	LG TALN series
67	Philips TD1316 Hybrid Tuner
68	Philips TUV1236D ATSC/NTSC dual in
69	Tena TNF 5335 and similar models
70	Samsung TCPN 2121P30A
71	Xceive xc2028/xc3028 tuner
72	Thomson FE6600
73	Samsung TCPG 6121P30A
75	Philips TEA5761 FM Radio
76	Xceive 5000 tuner
77	TCL tuner MF02GIP-5N-E
78	Philips FMD1216MEX MK3 Hybrid Tuner
79	Philips PAL/SECAM multi (FM1216 MK5)
80	Philips FQ1216LME MK3 PAL/SECAM w/active loopthrough
81	Partsnic (Daewoo) PTI-5NF05
82	Philips CU1216L
83	NXP TDA18271
84	Sony BTF-Pxn01Z
85	Philips FQ1236 MK5
86	Tena TNF5337 MFD
87	Xceive 4000 tuner
88	Xceive 5000C tuner
89	Sony BTF-PG472Z PAL/SECAM
90	Sony BTF-PK467Z NTSC-M-JP
91	Sony BTF-PB463Z NTSC-M

4.6.11 USBvision cards list

Card number	Card name	USB IDs
0	Xanboo	0a6f:0400
1	Belkin USB VideoBus II Adapter	050d:0106
2	Belkin Components USB VideoBus	050d:0207
3	Belkin USB VideoBus II	050d:0208
4	echoFX InterView Lite	0571:0002
5	USBGear USBG-V1 resp. HAMA USB	0573:0003
6	D-Link V100	0573:0400
7	X10 USB Camera	0573:2000
8	Hauppauge WinTV USB Live (PAL B/G)	0573:2d00

Continued on next page

Table 4.8 – continued from previous page

Card number	Card name	USB IDs
9	Hauppauge WinTV USB Live Pro (NTSC M/N)	0573:2d01
10	Zoran Co. PMD (Nogatech) AV-grabber Manhattan	0573:2101
11	Nogatech USB-TV (NTSC) FM	0573:4100
12	PNY USB-TV (NTSC) FM	0573:4110
13	PixelView PlayTv-USB PRO (PAL) FM	0573:4450
14	ZTV ZT-721 2.4GHz USB A/V Receiver	0573:4550
15	Hauppauge WinTV USB (NTSC M/N)	0573:4d00
16	Hauppauge WinTV USB (PAL B/G)	0573:4d01
17	Hauppauge WinTV USB (PAL I)	0573:4d02
18	Hauppauge WinTV USB (PAL/SECAM L)	0573:4d03
19	Hauppauge WinTV USB (PAL D/K)	0573:4d04
20	Hauppauge WinTV USB (NTSC FM)	0573:4d10
21	Hauppauge WinTV USB (PAL B/G FM)	0573:4d11
22	Hauppauge WinTV USB (PAL I FM)	0573:4d12
23	Hauppauge WinTV USB (PAL D/K FM)	0573:4d14
24	Hauppauge WinTV USB Pro (NTSC M/N)	0573:4d2a
25	Hauppauge WinTV USB Pro (NTSC M/N) V2	0573:4d2b
26	Hauppauge WinTV USB Pro (PAL/SECAM B/G/I/D/K/L)	0573:4d2c
27	Hauppauge WinTV USB Pro (NTSC M/N) V3	0573:4d20
28	Hauppauge WinTV USB Pro (PAL B/G)	0573:4d21
29	Hauppauge WinTV USB Pro (PAL I)	0573:4d22
30	Hauppauge WinTV USB Pro (PAL/SECAM L)	0573:4d23
31	Hauppauge WinTV USB Pro (PAL D/K)	0573:4d24
32	Hauppauge WinTV USB Pro (PAL/SECAM BGDK/I/L)	0573:4d25
33	Hauppauge WinTV USB Pro (PAL/SECAM BGDK/I/L) V2	0573:4d26
34	Hauppauge WinTV USB Pro (PAL B/G) V2	0573:4d27
35	Hauppauge WinTV USB Pro (PAL B/G,D/K)	0573:4d28
36	Hauppauge WinTV USB Pro (PAL I,D/K)	0573:4d29
37	Hauppauge WinTV USB Pro (NTSC M/N FM)	0573:4d30
38	Hauppauge WinTV USB Pro (PAL B/G FM)	0573:4d31
39	Hauppauge WinTV USB Pro (PAL I FM)	0573:4d32
40	Hauppauge WinTV USB Pro (PAL D/K FM)	0573:4d34
41	Hauppauge WinTV USB Pro (Temic PAL/SECAM B/G/I/D/K/L FM)	0573:4d35
42	Hauppauge WinTV USB Pro (Temic PAL B/G FM)	0573:4d36
43	Hauppauge WinTV USB Pro (PAL/SECAM B/G/I/D/K/L FM)	0573:4d37
44	Hauppauge WinTV USB Pro (NTSC M/N FM) V2	0573:4d38
45	Camtel Technology USB TV Genie Pro FM Model TVB330	0768:0006
46	Digital Video Creator I	07d0:0001
47	Global Village GV-007 (NTSC)	07d0:0002
48	Dazzle Fusion Model DVC-50 Rev 1 (NTSC)	07d0:0003
49	Dazzle Fusion Model DVC-80 Rev 1 (PAL)	07d0:0004
50	Dazzle Fusion Model DVC-90 Rev 1 (SECAM)	07d0:0005
51	Eskape Labs MyTV2Go	07f8:9104
52	Pinnacle Studio PCTV USB (PAL)	2304:010d
53	Pinnacle Studio PCTV USB (SECAM)	2304:0109
54	Pinnacle Studio PCTV USB (PAL) FM	2304:0110
55	Miro PCTV USB	2304:0111
56	Pinnacle Studio PCTV USB (NTSC) FM	2304:0112
57	Pinnacle Studio PCTV USB (PAL) FM V2	2304:0210
58	Pinnacle Studio PCTV USB (NTSC) FM V2	2304:0212
59	Pinnacle Studio PCTV USB (PAL) FM V3	2304:0214
60	Pinnacle Studio Linx Video input cable (NTSC)	2304:0300
61	Pinnacle Studio Linx Video input cable (PAL)	2304:0301

Continued on next page

Table 4.8 – continued from previous page

Card number	Card name	USB IDs
62	Pinnacle PCTV Bungee USB (PAL) FM	2304:0419
63	Hauppauge WinTv-USB	2400:4200
64	Pinnacle Studio PCTV USB (NTSC) FM V3	2304:0113
65	Nogatech USB MicroCam NTSC (NV3000N)	0573:3000
66	Nogatech USB MicroCam PAL (NV3001P)	0573:3001

4.6.12 The gspca cards list

The modules for the gspca webcam drivers are:

- gspca_main: main driver
- gspca_*driver*: subdriver module with *driver* as follows

driver	vend:prod	Device
spca501	0000:0000	MystFromOri Unknown Camera
spca508	0130:0130	Clone Digital Webcam 11043
se401	03e8:0004	Endpoints/AoxSE401
zc3xx	03f0:1b07	HP Premium Starter Cam
m5602	0402:5602	ALi Video Camera Controller
spca501	040a:0002	Kodak DVC-325
spca500	040a:0300	Kodak EZ200
zc3xx	041e:041e	Creative WebCam Live!
ov519	041e:4003	Video Blaster WebCam Go Plus
stv0680	041e:4007	Go Mini
spca500	041e:400a	Creative PC-CAM 300
sunplus	041e:400b	Creative PC-CAM 600
sunplus	041e:4012	PC-Cam350
sunplus	041e:4013	Creative Pccam750
zc3xx	041e:4017	Creative Webcam Mobile PD1090
spca508	041e:4018	Creative Webcam Vista (PD1100)
spca561	041e:401a	Creative Webcam Vista (PD1100)
zc3xx	041e:401c	Creative NX
spca505	041e:401d	Creative Webcam NX ULTRA
zc3xx	041e:401e	Creative Nx Pro
zc3xx	041e:401f	Creative Webcam Notebook PD1171
zc3xx	041e:4022	Webcam NX Pro
pac207	041e:4028	Creative Webcam Vista Plus
zc3xx	041e:4029	Creative WebCam Vista Pro
zc3xx	041e:4034	Creative Instant P0620
zc3xx	041e:4035	Creative Instant P0620D
zc3xx	041e:4036	Creative Live !
sq930x	041e:4038	Creative Joy-IT
zc3xx	041e:403a	Creative Nx Pro 2
spca561	041e:403b	Creative Webcam Vista (VF0010)
sq930x	041e:403c	Creative Live! Ultra
sq930x	041e:403d	Creative Live! Ultra for Notebooks
sq930x	041e:4041	Creative Live! Motion
zc3xx	041e:4051	Creative Live!Cam Notebook Pro (VF0250)
ov519	041e:4052	Creative Live! VISTA IM
zc3xx	041e:4053	Creative Live!Cam Video IM
vc032x	041e:405b	Creative Live! Cam Notebook Ultra (VC0130)
ov519	041e:405f	Creative Live! VISTA VF0330
ov519	041e:4060	Creative Live! VISTA VF0350

Continued on next page

Table 4.9 – continued from previous page

driver	vend:prod	Device
ov519	041e:4061	Creative Live! VISTA VF0400
ov519	041e:4064	Creative Live! VISTA VF0420
ov519	041e:4067	Creative Live! Cam Video IM (VF0350)
ov519	041e:4068	Creative Live! VISTA VF0470
sn9c2028	0458:7003	GeniusVideocam Live v2
spca561	0458:7004	Genius VideoCAM Express V2
sn9c2028	0458:7005	Genius Smart 300, version 2
sunplus	0458:7006	Genius Dsc 1.3 Smart
zc3xx	0458:7007	Genius VideoCam V2
zc3xx	0458:700c	Genius VideoCam V3
zc3xx	0458:700f	Genius VideoCam Web V2
sonixj	0458:7025	Genius Eye 311Q
sn9c20x	0458:7029	Genius Look 320s
sonixj	0458:702e	Genius Slim 310 NB
sn9c20x	0458:7045	Genius Look 1320 V2
sn9c20x	0458:704a	Genius Slim 1320
sn9c20x	0458:704c	Genius i-Look 1321
sn9c20x	045e:00f4	LifeCam VX-6000 (SN9C20x + OV9650)
sonixj	045e:00f5	MicroSoft VX3000
sonixj	045e:00f7	MicroSoft VX1000
ov519	045e:028c	Micro$oft xbox cam
kinect	045e:02ae	Xbox NUI Camera
kinect	045e:02bf	Kinect for Windows NUI Camera
spca561	0461:0815	Micro Innovations IC200 Webcam
sunplus	0461:0821	Fujifilm MV-1
zc3xx	0461:0a00	MicroInnovation WebCam320
stv06xx	046D:08F0	QuickCamMessenger
stv06xx	046D:08F5	QuickCamCommunicate
stv06xx	046D:08F6	QuickCamMessenger (new)
stv06xx	046d:0840	QuickCamExpress
stv06xx	046d:0850	LEGOcam / QuickCam Web
stv06xx	046d:0870	DexxaWebCam USB
spca500	046d:0890	Logitech QuickCam traveler
vc032x	046d:0892	Logitech Orbicam
vc032x	046d:0896	Logitech Orbicam
vc032x	046d:0897	Logitech QuickCam for Dell notebooks
zc3xx	046d:089d	Logitech QuickCam E2500
zc3xx	046d:08a0	Logitech QC IM
zc3xx	046d:08a1	Logitech QC IM 0x08A1 +sound
zc3xx	046d:08a2	Labtec Webcam Pro
zc3xx	046d:08a3	Logitech QC Chat
zc3xx	046d:08a6	Logitech QCim
zc3xx	046d:08a7	Logitech QuickCam Image
zc3xx	046d:08a9	Logitech Notebook Deluxe
zc3xx	046d:08aa	Labtec Webcam Notebook
zc3xx	046d:08ac	Logitech QuickCam Cool
zc3xx	046d:08ad	Logitech QCCommunicate STX
zc3xx	046d:08ae	Logitech QuickCam for Notebooks
zc3xx	046d:08af	Logitech QuickCam Cool
zc3xx	046d:08b9	Logitech QuickCam Express
zc3xx	046d:08d7	Logitech QCam STX
zc3xx	046d:08d8	Logitech Notebook Deluxe
zc3xx	046d:08d9	Logitech QuickCam IM/Connect

Continued on next page

Table 4.9 – continued from previous page

driver	vend:prod	Device
zc3xx	046d:08da	Logitech QuickCam Messenger
zc3xx	046d:08dd	Logitech QuickCam for Notebooks
spca500	046d:0900	Logitech Inc. ClickSmart 310
spca500	046d:0901	Logitech Inc. ClickSmart 510
sunplus	046d:0905	Logitech ClickSmart 820
tv8532	046d:0920	Logitech QuickCam Express
tv8532	046d:0921	Labtec Webcam
spca561	046d:0928	Logitech QC Express Etch2
spca561	046d:0929	Labtec Webcam Elch2
spca561	046d:092a	Logitech QC for Notebook
spca561	046d:092b	Labtec Webcam Plus
spca561	046d:092c	Logitech QC chat Elch2
spca561	046d:092d	Logitech QC Elch2
spca561	046d:092e	Logitech QC Elch2
spca561	046d:092f	Logitech QuickCam Express Plus
sunplus	046d:0960	Logitech ClickSmart 420
nw80x	046d:d001	Logitech QuickCam Pro (dark focus ring)
se401	0471:030b	PhilipsPCVC665K
sunplus	0471:0322	Philips DMVC1300K
zc3xx	0471:0325	Philips SPC 200 NC
zc3xx	0471:0326	Philips SPC 300 NC
sonixj	0471:0327	Philips SPC 600 NC
sonixj	0471:0328	Philips SPC 700 NC
zc3xx	0471:032d	Philips SPC 210 NC
zc3xx	0471:032e	Philips SPC 315 NC
sonixj	0471:0330	Philips SPC 710 NC
se401	047d:5001	Kensington67014
se401	047d:5002	Kensington6701(5/7)
se401	047d:5003	Kensington67016
spca501	0497:c001	Smile International
sunplus	04a5:3003	Benq DC 1300
sunplus	04a5:3008	Benq DC 1500
sunplus	04a5:300a	Benq DC 3410
spca500	04a5:300c	Benq DC 1016
benq	04a5:3035	Benq DC E300
vicam	04c1:009d	HomeConnect Webcam [vicam]
konica	04c8:0720	IntelYC 76
finepix	04cb:0104	Fujifilm FinePix 4800
finepix	04cb:0109	Fujifilm FinePix A202
finepix	04cb:010b	Fujifilm FinePix A203
finepix	04cb:010f	Fujifilm FinePix A204
finepix	04cb:0111	Fujifilm FinePix A205
finepix	04cb:0113	Fujifilm FinePix A210
finepix	04cb:0115	Fujifilm FinePix A303
finepix	04cb:0117	Fujifilm FinePix A310
finepix	04cb:0119	Fujifilm FinePix F401
finepix	04cb:011b	Fujifilm FinePix F402
finepix	04cb:011d	Fujifilm FinePix F410
finepix	04cb:0121	Fujifilm FinePix F601
finepix	04cb:0123	Fujifilm FinePix F700
finepix	04cb:0125	Fujifilm FinePix M603
finepix	04cb:0127	Fujifilm FinePix S300
finepix	04cb:0129	Fujifilm FinePix S304

Continued on next page

Table 4.9 – continued from previous page

driver	vend:prod	Device
finepix	04cb:012b	Fujifilm FinePix S500
finepix	04cb:012d	Fujifilm FinePix S602
finepix	04cb:012f	Fujifilm FinePix S700
finepix	04cb:0131	Fujifilm FinePix unknown model
finepix	04cb:013b	Fujifilm FinePix unknown model
finepix	04cb:013d	Fujifilm FinePix unknown model
finepix	04cb:013f	Fujifilm FinePix F420
sunplus	04f1:1001	JVC GC A50
spca561	04fc:0561	Flexcam 100
spca1528	04fc:1528	Sunplus MD80 clone
sunplus	04fc:500c	Sunplus CA500C
sunplus	04fc:504a	Aiptek Mini PenCam 1.3
sunplus	04fc:504b	Maxell MaxPocket LE 1.3
sunplus	04fc:5330	Digitrex 2110
sunplus	04fc:5360	Sunplus Generic
spca500	04fc:7333	PalmPixDC85
sunplus	04fc:ffff	Pure DigitalDakota
nw80x	0502:d001	DVC V6
spca501	0506:00df	3Com HomeConnect Lite
sunplus	052b:1507	Megapixel 5 Pretec DC-1007
sunplus	052b:1513	Megapix V4
sunplus	052b:1803	MegaImage VI
nw80x	052b:d001	EZCam Pro p35u
tv8532	0545:808b	Veo Stingray
tv8532	0545:8333	Veo Stingray
sunplus	0546:3155	Polaroid PDC3070
sunplus	0546:3191	Polaroid Ion 80
sunplus	0546:3273	Polaroid PDC2030
touptek	0547:6801	TTUCMOS08000KPB, AS MU800
dtcs033	0547:7303	Anchor Chips, Inc
ov519	054c:0154	Sonny toy4
ov519	054c:0155	Sonny toy5
cpia1	0553:0002	CPIA CPiA (version1) based cameras
stv0680	0553:0202	STV0680 Camera
zc3xx	055f:c005	Mustek Wcam300A
spca500	055f:c200	Mustek Gsmart 300
sunplus	055f:c211	Kowa Bs888e Microcamera
spca500	055f:c220	Gsmart Mini
sunplus	055f:c230	Mustek Digicam 330K
sunplus	055f:c232	Mustek MDC3500
sunplus	055f:c360	Mustek DV4000 Mpeg4
sunplus	055f:c420	Mustek gSmart Mini 2
sunplus	055f:c430	Mustek Gsmart LCD 2
sunplus	055f:c440	Mustek DV 3000
sunplus	055f:c520	Mustek gSmart Mini 3
sunplus	055f:c530	Mustek Gsmart LCD 3
sunplus	055f:c540	Gsmart D30
sunplus	055f:c630	Mustek MDC4000
sunplus	055f:c650	Mustek MDC5500Z
nw80x	055f:d001	Mustek Wcam 300 mini
zc3xx	055f:d003	Mustek WCam300A
zc3xx	055f:d004	Mustek WCam300 AN
conex	0572:0041	Creative Notebook cx11646

Continued on next page

Table 4.9 – continued from previous page

driver	vend:prod	Device
ov519	05a9:0511	Video Blaster WebCam 3/WebCam Plus, D-Link USB Digital Video Camera
ov519	05a9:0518	Creative WebCam
ov519	05a9:0519	OV519 Microphone
ov519	05a9:0530	OmniVision
ov534_9	05a9:1550	OmniVision VEHO Filmscanner
ov519	05a9:2800	OmniVision SuperCAM
ov519	05a9:4519	Webcam Classic
ov534_9	05a9:8065	OmniVision test kit ov538+ov9712
ov519	05a9:8519	OmniVision
ov519	05a9:a511	D-Link USB Digital Video Camera
ov519	05a9:a518	D-Link DSB-C310 Webcam
sunplus	05da:1018	Digital Dream Enigma 1.3
stk014	05e1:0893	Syntek DV4000
gl860	05e3:0503	Genesys Logic PC Camera
gl860	05e3:f191	Genesys Logic PC Camera
vicam	0602:1001	ViCam Webcam
spca561	060b:a001	Maxell Compact Pc PM3
zc3xx	0698:2003	CTX M730V built in
topro	06a2:0003	TP6800 PC Camera, CmoX CX0342 webcam
topro	06a2:6810	Creative Qmax
nw80x	06a5:0000	Typhoon Webcam 100 USB
nw80x	06a5:d001	Divio based webcams
nw80x	06a5:d800	Divio Chicony TwinkleCam, Trust SpaceCam
spca500	06bd:0404	Agfa CL20
spca500	06be:0800	Optimedia
nw80x	06be:d001	EZCam Pro p35u
sunplus	06d6:0031	Trust 610 LCD PowerC@m Zoom
sunplus	06d6:0041	Aashima Technology B.V.
spca506	06e1:a190	ADS Instant VCD
ov534	06f8:3002	Hercules Blog Webcam
ov534_9	06f8:3003	Hercules Dualpix HD Weblog
sonixj	06f8:3004	Hercules Classic Silver
sonixj	06f8:3008	Hercules Deluxe Optical Glass
pac7302	06f8:3009	Hercules Classic Link
pac7302	06f8:301b	Hercules Link
nw80x	0728:d001	AVerMedia Camguard
spca508	0733:0110	ViewQuest VQ110
spca501	0733:0401	Intel Create and Share
spca501	0733:0402	ViewQuest M318B
spca505	0733:0430	Intel PC Camera Pro
sunplus	0733:1311	Digital Dream Epsilon 1.3
sunplus	0733:1314	Mercury 2.1MEG Deluxe Classic Cam
sunplus	0733:2211	Jenoptik jdc 21 LCD
sunplus	0733:2221	Mercury Digital Pro 3.1p
sunplus	0733:3261	Concord 3045 spca536a
sunplus	0733:3281	Cyberpix S550V
spca506	0734:043b	3DeMon USB Capture aka
cpia1	0813:0001	QX3 camera
ov519	0813:0002	Dual Mode USB Camera Plus
spca500	084d:0003	D-Link DSC-350
spca500	08ca:0103	Aiptek PocketDV
sunplus	08ca:0104	Aiptek PocketDVII 1.3
sunplus	08ca:0106	Aiptek Pocket DV3100+

Continued on next page

Table 4.9 – continued from previous page

driver	vend:prod	Device
mr97310a	08ca:0110	Trust Spyc@m 100
mr97310a	08ca:0111	Aiptek PenCam VGA+
sunplus	08ca:2008	Aiptek Mini PenCam 2 M
sunplus	08ca:2010	Aiptek PocketCam 3M
sunplus	08ca:2016	Aiptek PocketCam 2 Mega
sunplus	08ca:2018	Aiptek Pencam SD 2M
sunplus	08ca:2020	Aiptek Slim 3000F
sunplus	08ca:2022	Aiptek Slim 3200
sunplus	08ca:2024	Aiptek DV3500 Mpeg4
sunplus	08ca:2028	Aiptek PocketCam4M
sunplus	08ca:2040	Aiptek PocketDV4100M
sunplus	08ca:2042	Aiptek PocketDV5100
sunplus	08ca:2050	Medion MD 41437
sunplus	08ca:2060	Aiptek PocketDV5300
tv8532	0923:010f	ICM532 cams
mr97310a	093a:010e	All known CIF cams with this ID
mr97310a	093a:010f	All known VGA cams with this ID
mars	093a:050f	Mars-Semi Pc-Camera
pac207	093a:2460	Qtec Webcam 100
pac207	093a:2461	HP Webcam
pac207	093a:2463	Philips SPC 220 NC
pac207	093a:2464	Labtec Webcam 1200
pac207	093a:2468	Webcam WB-1400T
pac207	093a:2470	Genius GF112
pac207	093a:2471	Genius VideoCam ge111
pac207	093a:2472	Genius VideoCam ge110
pac207	093a:2474	Genius iLook 111
pac207	093a:2476	Genius e-Messenger 112
pac7311	093a:2600	PAC7311 Typhoon
pac7311	093a:2601	Philips SPC 610 NC
pac7311	093a:2603	Philips SPC 500 NC
pac7311	093a:2608	Trust WB-3300p
pac7311	093a:260e	Gigaware VGA PC Camera, Trust WB-3350p, SIGMA cam 2350
pac7311	093a:260f	SnakeCam
pac7302	093a:2620	Apollo AC-905
pac7302	093a:2621	PAC731x
pac7302	093a:2622	Genius Eye 312
pac7302	093a:2623	Pixart Imaging, Inc.
pac7302	093a:2624	PAC7302
pac7302	093a:2625	Genius iSlim 310
pac7302	093a:2626	Labtec 2200
pac7302	093a:2627	Genius FaceCam 300
pac7302	093a:2628	Genius iLook 300
pac7302	093a:2629	Genious iSlim 300
pac7302	093a:262a	Webcam 300k
pac7302	093a:262c	Philips SPC 230 NC
jl2005bcd	0979:0227	Various brands, 19 known cameras supported
jeilinj	0979:0270	Sakar 57379
jeilinj	0979:0280	Sportscam DV15, Sakar 57379
zc3xx	0ac8:0301	Web Camera
zc3xx	0ac8:0302	Z-star Vimicro zc0302
vc032x	0ac8:0321	Vimicro generic vc0321
vc032x	0ac8:0323	Vimicro Vc0323

Continued on next page

Table 4.9 – continued from previous page

driver	vend:prod	Device
vc032x	0ac8:0328	A4Tech PK-130MG
zc3xx	0ac8:301b	Z-Star zc301b
zc3xx	0ac8:303b	Vimicro 0x303b
zc3xx	0ac8:305b	Z-star Vimicro zc0305b
zc3xx	0ac8:307b	PC Camera (ZS0211)
vc032x	0ac8:c001	Sony embedded vimicro
vc032x	0ac8:c002	Sony embedded vimicro
vc032x	0ac8:c301	Samsung Q1 Ultra Premium
spca508	0af9:0010	Hama USB Sightcam 100
spca508	0af9:0011	Hama USB Sightcam 100
ov519	0b62:0059	iBOT2 Webcam
sonixb	0c45:6001	Genius VideoCAM NB
sonixb	0c45:6005	Microdia Sweex Mini Webcam
sonixb	0c45:6007	Sonix sn9c101 + Tas5110D
sonixb	0c45:6009	spcaCam@120
sonixb	0c45:600d	spcaCam@120
sonixb	0c45:6011	Microdia PC Camera (SN9C102)
sonixb	0c45:6019	Generic Sonix OV7630
sonixb	0c45:6024	Generic Sonix Tas5130c
sonixb	0c45:6025	Xcam Shanga
sonixb	0c45:6027	GeniusEye 310
sonixb	0c45:6028	Sonix Btc Pc380
sonixb	0c45:6029	spcaCam@150
sonixb	0c45:602a	Meade ETX-105EC Camera
sonixb	0c45:602c	Generic Sonix OV7630
sonixb	0c45:602d	LIC-200 LG
sonixb	0c45:602e	Genius VideoCam Messenger
sonixj	0c45:6040	Speed NVC 350K
sonixj	0c45:607c	Sonix sn9c102p Hv7131R
sonixb	0c45:6083	VideoCAM Look
sonixb	0c45:608c	VideoCAM Look
sonixb	0c45:608f	PC Camera (SN9C103 + OV7630)
sonixb	0c45:60a8	VideoCAM Look
sonixb	0c45:60aa	VideoCAM Look
sonixb	0c45:60af	VideoCAM Look
sonixb	0c45:60b0	Genius VideoCam Look
sonixj	0c45:60c0	Sangha Sn535
sonixj	0c45:60ce	USB-PC-Camera-168 (TALK-5067)
sonixj	0c45:60ec	SN9C105+MO4000
sonixj	0c45:60fb	Surfer NoName
sonixj	0c45:60fc	LG-LIC300
sonixj	0c45:60fe	Microdia Audio
sonixj	0c45:6100	PC Camera (SN9C128)
sonixj	0c45:6102	PC Camera (SN9C128)
sonixj	0c45:610a	PC Camera (SN9C128)
sonixj	0c45:610b	PC Camera (SN9C128)
sonixj	0c45:610c	PC Camera (SN9C128)
sonixj	0c45:610e	PC Camera (SN9C128)
sonixj	0c45:6128	Microdia/Sonix SNP325
sonixj	0c45:612a	Avant Camera
sonixj	0c45:612b	Speed-Link REFLECT2
sonixj	0c45:612c	Typhoon Rasy Cam 1.3MPix
sonixj	0c45:612e	PC Camera (SN9C110)

Continued on next page

Table 4.9 – continued from previous page

driver	vend:prod	Device
sonixj	0c45:6130	Sonix Pccam
sonixj	0c45:6138	Sn9c120 Mo4000
sonixj	0c45:613a	Microdia Sonix PC Camera
sonixj	0c45:613b	Surfer SN-206
sonixj	0c45:613c	Sonix Pccam168
sonixj	0c45:613e	PC Camera (SN9C120)
sonixj	0c45:6142	Hama PC-Webcam AC-150
sonixj	0c45:6143	Sonix Pccam168
sonixj	0c45:6148	Digitus DA-70811/ZSMC USB PC Camera ZS211/Microdia
sonixj	0c45:614a	Frontech E-Ccam (JIL-2225)
sn9c20x	0c45:6240	PC Camera (SN9C201 + MT9M001)
sn9c20x	0c45:6242	PC Camera (SN9C201 + MT9M111)
sn9c20x	0c45:6248	PC Camera (SN9C201 + OV9655)
sn9c20x	0c45:624c	PC Camera (SN9C201 + MT9M112)
sn9c20x	0c45:624e	PC Camera (SN9C201 + SOI968)
sn9c20x	0c45:624f	PC Camera (SN9C201 + OV9650)
sn9c20x	0c45:6251	PC Camera (SN9C201 + OV9650)
sn9c20x	0c45:6253	PC Camera (SN9C201 + OV9650)
sn9c20x	0c45:6260	PC Camera (SN9C201 + OV7670)
sn9c20x	0c45:6270	PC Camera (SN9C201 + MT9V011/MT9V111/MT9V112)
sn9c20x	0c45:627b	PC Camera (SN9C201 + OV7660)
sn9c20x	0c45:627c	PC Camera (SN9C201 + HV7131R)
sn9c20x	0c45:627f	PC Camera (SN9C201 + OV9650)
sn9c20x	0c45:6280	PC Camera (SN9C202 + MT9M001)
sn9c20x	0c45:6282	PC Camera (SN9C202 + MT9M111)
sn9c20x	0c45:6288	PC Camera (SN9C202 + OV9655)
sn9c20x	0c45:628c	PC Camera (SN9C201 + MT9M112)
sn9c20x	0c45:628e	PC Camera (SN9C202 + SOI968)
sn9c20x	0c45:628f	PC Camera (SN9C202 + OV9650)
sn9c20x	0c45:62a0	PC Camera (SN9C202 + OV7670)
sn9c20x	0c45:62b0	PC Camera (SN9C202 + MT9V011/MT9V111/MT9V112)
sn9c20x	0c45:62b3	PC Camera (SN9C202 + OV9655)
sn9c20x	0c45:62bb	PC Camera (SN9C202 + OV7660)
sn9c20x	0c45:62bc	PC Camera (SN9C202 + HV7131R)
sn9c2028	0c45:8001	Wild Planet Digital Spy Camera
sn9c2028	0c45:8003	Sakar #11199, #6637x, #67480 keychain cams
sn9c2028	0c45:8008	Mini-Shotz ms-350
sn9c2028	0c45:800a	Vivitar Vivicam 3350B
sunplus	0d64:0303	Sunplus FashionCam DXG
ov519	0e96:c001	TRUST 380 USB2 SPACEC@M
etoms	102c:6151	Qcam Sangha CIF
etoms	102c:6251	Qcam xxxxxx VGA
ov519	1046:9967	W9967CF/W9968CF WebCam IC, Video Blaster WebCam Go
zc3xx	10fd:0128	Typhoon Webshot II USB 300k 0x0128
spca561	10fd:7e50	FlyCam Usb 100
zc3xx	10fd:804d	Typhoon Webshot II Webcam [zc0301]
zc3xx	10fd:8050	Typhoon Webshot II USB 300k
ov534	1415:2000	Sony HD Eye for PS3 (SLEH 00201)
pac207	145f:013a	Trust WB-1300N
pac7302	145f:013c	Trust
sn9c20x	145f:013d	Trust WB-3600R
vc032x	15b8:6001	HP 2.0 Megapixel
vc032x	15b8:6002	HP 2.0 Megapixel rz406aa

Continued on next page

Table 4.9 – continued from previous page

driver	vend:prod	Device
stk1135	174f:6a31	ASUSlaptop, MT9M112 sensor
spca501	1776:501c	Arowana 300K CMOS Camera
t613	17a1:0128	TASCORP JPEG Webcam, NGS Cyclops
vc032x	17ef:4802	Lenovo Vc0323+MI1310_SOC
pac7302	1ae7:2001	SpeedLinkSnappy Mic SL-6825-SBK
pac207	2001:f115	D-Link DSB-C120
sq905c	2770:9050	Disney pix micro (CIF)
sq905c	2770:9051	Lego Bionicle
sq905c	2770:9052	Disney pix micro 2 (VGA)
sq905c	2770:905c	All 11 known cameras with this ID
sq905	2770:9120	All 24 known cameras with this ID
sq905c	2770:913d	All 4 known cameras with this ID
sq930x	2770:930b	Sweex Motion Tracking / I-Tec iCam Tracer
sq930x	2770:930c	Trust WB-3500T / NSG Robbie 2.0
spca500	2899:012c	Toptro Industrial
ov519	8020:ef04	ov519
spca508	8086:0110	Intel Easy PC Camera
spca500	8086:0630	Intel Pocket PC Camera
spca506	99fa:8988	Grandtec V.cap
sn9c20x	a168:0610	Dino-Lite Digital Microscope (SN9C201 + HV7131R)
sn9c20x	a168:0611	Dino-Lite Digital Microscope (SN9C201 + HV7131R)
sn9c20x	a168:0613	Dino-Lite Digital Microscope (SN9C201 + HV7131R)
sn9c20x	a168:0614	Dino-Lite Digital Microscope (SN9C201 + MT9M111)
sn9c20x	a168:0615	Dino-Lite Digital Microscope (SN9C201 + MT9M111)
sn9c20x	a168:0617	Dino-Lite Digital Microscope (SN9C201 + MT9M111)
sn9c20x	a168:0618	Dino-Lite Digital Microscope (SN9C201 + HV7131R)
spca561	abcd:cdee	Petcam

4.7 The bttv driver

4.7.1 Release notes for bttv

You'll need at least these config options for bttv:

```
CONFIG_I2C=m
CONFIG_I2C_ALGOBIT=m
CONFIG_VIDEO_DEV=m
```

The latest bttv version is available from http://bytesex.org/bttv/

4.7.2 Make bttv work with your card

Just try "modprobe bttv" and see if that works.

If it doesn't bttv likely could not autodetect your card and needs some insmod options. The most important insmod option for bttv is "card=n" to select the correct card type. If you get video but no sound you've very likely specified the wrong (or no) card type. A list of supported cards is in CARDLIST.bttv

If bttv takes very long to load (happens sometimes with the cheap cards which have no tuner), try adding this to your modules.conf:

```
options i2c-algo-bit bit_test=1
```

For the WinTV/PVR you need one firmware file from the driver CD: hcwamc.rbf. The file is in the pvr45xxx.exe archive (self-extracting zip file, unzip can unpack it). Put it into the /etc/pvr directory or use the firm_altera=<path> insmod option to point the driver to the location of the file.

If your card isn't listed in CARDLIST.bttv or if you have trouble making audio work, you should read the Sound-FAQ.

4.7.3 Autodetecting cards

bttv uses the PCI Subsystem ID to autodetect the card type. lspci lists the Subsystem ID in the second line, looks like this:

```
00:0a.0 Multimedia video controller: Brooktree Corporation Bt878 (rev 02)
        Subsystem: Hauppauge computer works Inc. WinTV/GO
        Flags: bus master, medium devsel, latency 32, IRQ 5
        Memory at e2000000 (32-bit, prefetchable) [size=4K]
```

only bt878-based cards can have a subsystem ID (which does not mean that every card really has one). bt848 cards can't have a Subsystem ID and therefore can't be autodetected. There is a list with the ID's in bttv-cards.c (in case you are intrested or want to mail patches with updates).

4.7.4 Still doesn't work?

I do NOT have a lab with 30+ different grabber boards and a PAL/NTSC/SECAM test signal generator at home, so I often can't reproduce your problems. This makes debugging very difficult for me. If you have some knowledge and spare time, please try to fix this yourself (patches very welcome of course...) You know: The linux slogan is "Do it yourself".

There is a mailing list at http://vger.kernel.org/vger-lists.html#linux-media

If you have trouble with some specific TV card, try to ask there instead of mailing me directly. The chance that someone with the same card listens there is much higher...

For problems with sound: There are a lot of different systems used for TV sound all over the world. And there are also different chips which decode the audio signal. Reports about sound problems ("stereo does'nt work") are pretty useless unless you include some details about your hardware and the TV sound scheme used in your country (or at least the country you are living in).

4.7.5 Modprobe options

Note: "modinfo <module>" prints various information about a kernel module, among them a complete and up-to-date list of insmod options. This list tends to be outdated because it is updated manually ...

---

bttv.o

```
the bt848/878 (grabber chip) driver

insmod args:
        card=n          card type, see CARDLIST for a list.
        tuner=n         tuner type, see CARDLIST for a list.
        radio=0/1       card supports radio
        pll=0/1/2       pll settings
                0: don't use PLL
                1: 28 MHz crystal installed
                2: 35 MHz crystal installed

        triton1=0/1     for Triton1 (+others) compatibility
        vsfx=0/1        yet another chipset bug compatibility bit
```

```
                        see README.quirks for details on these two.

        bigendian=n     Set the endianness of the gfx framebuffer.
                        Default is native endian.
        fieldnr=0/1     Count fields.  Some TV descrambling software
                        needs this, for others it only generates
                        50 useless IRQs/sec.  default is 0 (off).
        autoload=0/1    autoload helper modules (tuner, audio).
                        default is 1 (on).
        bttv_verbose=0/1/2  verbose level (at insmod time, while
                        looking at the hardware).  default is 1.
        bttv_debug=0/1  debug messages (for capture).
                        default is 0 (off).
        irq_debug=0/1   irq handler debug messages.
                        default is 0 (off).
        gbuffers=2-32   number of capture buffers for mmap'ed capture.
                        default is 4.
        gbufsize=       size of capture buffers. default and
                        maximum value is 0x208000 (~2MB)
        no_overlay=0    Enable overlay on broken hardware.  There
                        are some chipsets (SIS for example) which
                        are known to have problems with the PCI DMA
                        push used by bttv.  bttv will disable overlay
                        by default on this hardware to avoid crashes.
                        With this insmod option you can override this.
        no_overlay=1    Disable overlay. It should be used by broken
                        hardware that doesn't support PCI2PCI direct
                        transfers.
        automute=0/1    Automatically mutes the sound if there is
                        no TV signal, on by default.  You might try
                        to disable this if you have bad input signal
                        quality which leading to unwanted sound
                        dropouts.
        chroma_agc=0/1  AGC of chroma signal, off by default.
        adc_crush=0/1   Luminance ADC crush, on by default.
        i2c_udelay=     Allow reduce I2C speed. Default is 5 usecs
                        (meaning 66,67 Kbps). The default is the
                        maximum supported speed by kernel bitbang
                        algorithm. You may use lower numbers, if I2C
                        messages are lost (16 is known to work on
                        all supported cards).

        bttv_gpio=0/1
        gpiomask=
        audioall=
        audiomux=
                        See Sound-FAQ for a detailed description.

remap, card, radio and pll accept up to four comma-separated arguments
(for multiple boards).
```

tuner.o

```
The tuner driver.  You need this unless you want to use only
with a camera or external tuner ...

insmod args:
        debug=1         print some debug info to the syslog
        type=n          type of the tuner chip. n as follows:
                        see CARDLIST for a complete list.
        pal=[bdgil]     select PAL variant (used for some tuners
                        only, important for the audio carrier).
```

tvaudio.o

```
new, experimental module which is supported to provide a single
driver for all simple i2c audio control chips (tda/tea*).

insmod args:
        tda8425  = 1    enable/disable the support for the
        tda9840  = 1    various chips.
        tda9850  = 1    The tea6300 can't be autodetected and is
        tda9855  = 1    therefore off by default, if you have
        tda9873  = 1    this one on your card (STB uses these)
        tda9874a = 1    you have to enable it explicitly.
        tea6300  = 0    The two tda985x chips use the same i2c
        tea6420  = 1    address and can't be disturgished from
        pic16c54 = 1    each other, you might have to disable
                        the wrong one.
        debug = 1       print debug messages

insmod args for tda9874a:
        tda9874a_SIF=1/2        select sound IF input pin (1 or 2)
                                (default is pin 1)
        tda9874a_AMSEL=0/1      auto-mute select for NICAM (default=0)
                                Please read note 3 below!
        tda9874a_STD=n          select TV sound standard (0..8):
                                0 - A2, B/G
                                1 - A2, M (Korea)
                                2 - A2, D/K (1)
                                3 - A2, D/K (2)
                                4 - A2, D/K (3)
                                5 - NICAM, I
                                6 - NICAM, B/G
                                7 - NICAM, D/K (default)
                                8 - NICAM, L

Note 1: tda9874a supports both tda9874h (old) and tda9874a (new) chips.
Note 2: tda9874h/a and tda9875 (which is supported separately by
tda9875.o) use the same i2c address so both modules should not be
used at the same time.
Note 3: Using tda9874a_AMSEL option depends on your TV card design!
        AMSEL=0: auto-mute will switch between NICAM sound
                 and the sound on 1st carrier (i.e. FM mono or AM).
        AMSEL=1: auto-mute will switch between NICAM sound
                 and the analog mono input (MONOIN pin).
If tda9874a decoder on your card has MONOIN pin not connected, then
use only tda9874_AMSEL=0 or don't specify this option at all.
For example:
  card=65 (FlyVideo 2000S) - set AMSEL=1 or AMSEL=0
  card=72 (Prolink PV-BT878P rev.9B) - set AMSEL=0 only
```

msp3400.o

```
The driver for the msp34xx sound processor chips. If you have a
stereo card, you probably want to insmod this one.

insmod args:
        debug=1/2       print some debug info to the syslog,
                        2 is more verbose.
        simple=1        Use the "short programming" method.  Newer
                        msp34xx versions support this.  You need this
                        for dbx stereo.  Default is on if supported by
                        the chip.
        once=1          Don't check the TV-stations Audio mode
                        every few seconds, but only once after
```

```
                        channel switches.
        amsound=1       Audio carrier is AM/NICAM at 6.5 Mhz.  This
                        should improve things for french people, the
                        carrier autoscan seems to work with FM only...
```

tea6300.o - OBSOLETE (use tvaudio instead)

```
The driver for the tea6300 fader chip.  If you have a stereo
card and the msp3400.o doesn't work, you might want to try this
one.  This chip is seen on most STB TV/FM cards (usually from
Gateway OEM sold surplus on auction sites).

insmod args:
        debug=1         print some debug info to the syslog.
```

tda8425.o - OBSOLETE (use tvaudio instead)

```
The driver for the tda8425 fader chip.  This driver used to be
part of bttv.c, so if your sound used to work but does not
anymore, try loading this module.

insmod args:
        debug=1         print some debug info to the syslog.
```

tda985x.o - OBSOLETE (use tvaudio instead)

```
The driver for the tda9850/55 audio chips.

insmod args:
        debug=1         print some debug info to the syslog.
        chip=9850/9855  set the chip type.
```

4.7.6 If the box freezes hard with bttv

It might be a bttv driver bug. It also might be bad hardware. It also might be something else ...

Just mailing me “bttv freezes” isn’t going to help much. This README has a few hints how you can help to pin down the problem.

bttv bugs

If some version works and another doesn’t it is likely to be a driver bug. It is very helpful if you can tell where exactly it broke (i.e. the last working and the first broken version).

With a hard freeze you probably doesn’t find anything in the logfiles. The only way to capture any kernel messages is to hook up a serial console and let some terminal application log the messages. /me uses screen. See Documentation/admin-guide/serial-console.rst for details on setting up a serial console.

Read Documentation/admin-guide/oops-tracing.rst to learn how to get any useful information out of a register+stack dump printed by the kernel on protection faults (so-called “kernel oops”).

If you run into some kind of deadlock, you can try to dump a call trace for each process using sysrq-t (see Documentation/sysrq.txt). This way it is possible to figure where *exactly* some process in “D” state is stuck.

I’ve seen reports that bttv 0.7.x crashes whereas 0.8.x works rock solid for some people. Thus probably a small buglet left somewhere in bttv 0.7.x. I have no idea where exactly, it works stable for me and a lot of other people. But in case you have problems with the 0.7.x versions you can give 0.8.x a try ...

hardware bugs

Some hardware can't deal with PCI-PCI transfers (i.e. grabber => vga). Sometimes problems show up with bttv just because of the high load on the PCI bus. The bt848/878 chips have a few workarounds for known incompatibilities, see README.quirks.

Some folks report that increasing the pci latency helps too, althrought I'm not sure whenever this really fixes the problems or only makes it less likely to happen. Both bttv and btaudio have a insmod option to set the PCI latency of the device.

Some mainboard have problems to deal correctly with multiple devices doing DMA at the same time. bttv + ide seems to cause this sometimes, if this is the case you likely see freezes only with video and hard disk access at the same time. Updating the IDE driver to get the latest and greatest workarounds for hardware bugs might fix these problems.

other

If you use some binary-only yunk (like nvidia module) try to reproduce the problem without.

IRQ sharing is known to cause problems in some cases. It works just fine in theory and many configurations. Neverless it might be worth a try to shuffle around the PCI cards to give bttv another IRQ or make it share the IRQ with some other piece of hardware. IRQ sharing with VGA cards seems to cause trouble sometimes. I've also seen funny effects with bttv sharing the IRQ with the ACPI bridge (and apci-enabled kernel).

4.7.7 Bttv quirks

Below is what the bt878 data book says about the PCI bug compatibility modes of the bt878 chip.

The triton1 insmod option sets the EN_TBFX bit in the control register. The vsfx insmod option does the same for EN_VSFX bit. If you have stability problems you can try if one of these options makes your box work solid.

drivers/pci/quirks.c knows about these issues, this way these bits are enabled automagically for known-buggy chipsets (look at the kernel messages, bttv tells you).

Normal PCI Mode

The PCI REQ signal is the logical-or of the incoming function requests. The inter-nal GNT[0:1] signals are gated asynchronously with GNT and demultiplexed by the audio request signal. Thus the arbiter defaults to the video function at power-up and parks there during no requests for bus access. This is desirable since the video will request the bus more often. However, the audio will have highest bus access priority. Thus the audio will have first access to the bus even when issuing a request after the video request but before the PCI external arbiter has granted access to the Bt879. Neither function can preempt the other once on the bus. The duration to empty the entire video PCI FIFO onto the PCI bus is very short compared to the bus access latency the audio PCI FIFO can tolerate.

430FX Compatibility Mode

When using the 430FX PCI, the following rules will ensure compatibility:

1. Deassert REQ at the same time as asserting FRAME.
2. Do not reassert REQ to request another bus transaction until after finish-ing the previous transaction.

Since the individual bus masters do not have direct control of REQ, a simple logical-or of video and audio requests would violate the rules. Thus, both the arbiter and the initiator contain 430FX compatibility mode logic. To enable 430FX mode, set the EN_TBFX bit as indicated in Device Control Register on page 104.

When EN_TBFX is enabled, the arbiter ensures that the two compatibility rules are satisfied. Before GNT is asserted by the PCI arbiter, this internal arbiter may still logical-or the two requests. However, once the GNT is issued, this arbiter must lock in its decision and now route only the granted request to the REQ pin. The arbiter decision lock happens regardless of the state of FRAME because it does not know when FRAME will be asserted (typically - each initiator will assert FRAME on the cycle following GNT). When FRAME is asserted, it is the initiator s responsibility to remove its request at the same time. It is the arbiters responsibility to allow this request to flow through to REQ and not allow the other request to hold REQ asserted. The decision lock may be removed at the end of the transaction: for example, when the bus is idle (FRAME and IRDY). The arbiter decision may then continue asynchronously until GNT is again asserted.

Interfacing with Non-PCI 2.1 Compliant Core Logic

A small percentage of core logic devices may start a bus transaction during the same cycle that GNT is de-asserted. This is non PCI 2.1 compliant. To ensure compatibility when using PCs with these PCI controllers, the EN_VSFX bit must be enabled (refer to Device Control Register on page 104). When in this mode, the arbiter does not pass GNT to the internal functions unless REQ is asserted. This prevents a bus transaction from starting the same cycle as GNT is de-asserted. This also has the side effect of not being able to take advantage of bus parking, thus lowering arbitration performance. The Bt879 drivers must query for these non-compliant devices, and set the EN_VSFX bit only if required.

4.7.8 bttv and sound mini howto

There are a lot of different bt848/849/878/879 based boards available. Making video work often is not a big deal, because this is handled completely by the bt8xx chip, which is common on all boards. But sound is handled in slightly different ways on each board.

To handle the grabber boards correctly, there is a array tvcards[] in bttv-cards.c, which holds the information required for each board. Sound will work only, if the correct entry is used (for video it often makes no difference). The bttv driver prints a line to the kernel log, telling which card type is used. Like this one:

```
bttv0: model: BT848(Hauppauge old) [autodetected]
```

You should verify this is correct. If it isn't, you have to pass the correct board type as insmod argument, "insmod bttv card=2" for example. The file CARDLIST has a list of valid arguments for card. If your card isn't listed there, you might check the source code for new entries which are not listed yet. If there isn't one for your card, you can check if one of the existing entries does work for you (just trial and error...).

Some boards have an extra processor for sound to do stereo decoding and other nice features. The msp34xx chips are used by Hauppauge for example. If your board has one, you might have to load a helper module like msp3400.o to make sound work. If there isn't one for the chip used on your board: Bad luck. Start writing a new one. Well, you might want to check the video4linux mailing list archive first...

Of course you need a correctly installed soundcard unless you have the speakers connected directly to the grabber board. Hint: check the mixer settings too. ALSA for example has everything muted by default.

How sound works in detail

Still doesn't work? Looks like some driver hacking is required. Below is a do-it-yourself description for you.

The bt8xx chips have 32 general purpose pins, and registers to control these pins. One register is the output enable register (BT848_GPIO_OUT_EN), it says which pins are actively driven by the bt848 chip. Another one is the data register (BT848_GPIO_DATA), where you can get/set the status if these pins. They can be used for input and output.

Most grabber board vendors use these pins to control an external chip which does the sound routing. But every board is a little different. These pins are also used by some companies to drive remote control receiver chips. Some boards use the i2c bus instead of the gpio pins to connect the mux chip.

As mentioned above, there is a array which holds the required information for each known board. You basically have to create a new line for your board. The important fields are these two:

```
struct tvcard
{
        [ ... ]
        u32 gpiomask;
        u32 audiomux[6]; /* Tuner, Radio, external, internal, mute, stereo */
};
```

gpiomask specifies which pins are used to control the audio mux chip. The corresponding bits in the output enable register (BT848_GPIO_OUT_EN) will be set as these pins must be driven by the bt848 chip.

The audiomux[] array holds the data values for the different inputs (i.e. which pins must be high/low for tuner/mute/...). This will be written to the data register (BT848_GPIO_DATA) to switch the audio mux.

What you have to do is figure out the correct values for gpiomask and the audiomux array. If you have Windows and the drivers four your card installed, you might to check out if you can read these registers values used by the windows driver. A tool to do this is available from ftp://telepresence.dmem.strath.ac.uk/pub/bt848/winutil, but it doesn't work with bt878 boards according to some reports I received. Another one with bt878 support is available from http://btwincap.sourceforge.net/Files/btspy2.00.zip

You might also dig around in the *.ini files of the Windows applications. You can have a look at the board to see which of the gpio pins are connected at all and then start trial-and-error ...

Starting with release 0.7.41 bttv has a number of insmod options to make the gpio debugging easier:

```
bttv_gpio=0/1           enable/disable gpio debug messages
gpiomask=n              set the gpiomask value
audiomux=i,j,...        set the values of the audiomux array
audioall=a              set the values of the audiomux array (one
                        value for all array elements, useful to check
                        out which effect the particular value has).
```

The messages printed with bttv_gpio=1 look like this:

```
bttv0: gpio: en=00000027, out=00000024 in=00ffffd8 [audio: off]

en  =   output _en_able register (BT848_GPIO_OUT_EN)
out =   _out_put bits of the data register (BT848_GPIO_DATA),
        i.e. BT848_GPIO_DATA & BT848_GPIO_OUT_EN
in  =   _in_put bits of the data register,
        i.e. BT848_GPIO_DATA & ~BT848_GPIO_OUT_EN
```

Other elements of the tvcards array

If you are trying to make a new card work you might find it useful to know what the other elements in the tvcards array are good for:

```
video_inputs    - # of video inputs the card has
audio_inputs    - historical cruft, not used any more.
tuner           - which input is the tuner
svhs            - which input is svhs (all others are labeled composite)
muxsel          - video mux, input->registervalue mapping
pll             - same as pll= insmod option
tuner_type      - same as tuner= insmod option
*_modulename    - hint whenever some card needs this or that audio
                module loaded to work properly.
```

```
has_radio       - whenever this TV card has a radio tuner.
no_msp34xx      - "1" disables loading of msp3400.o module
no_tda9875      - "1" disables loading of tda9875.o module
needs_tvaudio   - set to "1" to load tvaudio.o module
```

If some config item is specified both from the tvcards array and as insmod option, the insmod option takes precedence.

4.7.9 Cards

Note:

For a more updated list, please check https://linuxtv.org/wiki/index.php/Hardware_Device_Information

Supported cards: Bt848/Bt848a/Bt849/Bt878/Bt879 cards

All cards with Bt848/Bt848a/Bt849/Bt878/Bt879 and normal Composite/S-VHS inputs are supported. Teletext and Intercast support (PAL only) for ALL cards via VBI sample decoding in software.

Some cards with additional multiplexing of inputs or other additional fancy chips are only partially supported (unless specifications by the card manufacturer are given). When a card is listed here it isn't necessarily fully supported.

All other cards only differ by additional components as tuners, sound decoders, EEPROMs, teletext decoders ...

MATRIX Vision

MV-Delta - Bt848A - 4 Composite inputs, 1 S-VHS input (shared with 4th composite) - EEPROM

http://www.matrix-vision.de/

This card has no tuner but supports all 4 composite (1 shared with an S-VHS input) of the Bt848A. Very nice card if you only have satellite TV but several tuners connected to the card via composite.

Many thanks to Matrix-Vision for giving us 2 cards for free which made Bt848a/Bt849 single crystal operation support possible!!!

Miro/Pinnacle PCTV

- Bt848 some (all??) come with 2 crystals for PAL/SECAM and NTSC
- PAL, SECAM or NTSC TV tuner (Philips or TEMIC)
- MSP34xx sound decoder on add on board decoder is supported but AFAIK does not yet work (other sound MUX setting in GPIO port needed??? somebody who fixed this???)
- 1 tuner, 1 composite and 1 S-VHS input
- tuner type is autodetected

http://www.miro.de/ http://www.miro.com/

Many thanks for the free card which made first NTSC support possible back in 1997!

Hauppauge Win/TV pci

There are many different versions of the Hauppauge cards with different tuners (TV+Radio ...), teletext decoders. Note that even cards with same model numbers have (depending on the revision) different chips on it.

- Bt848 (and others but always in 2 crystal operation???) newer cards have a Bt878
- PAL, SECAM, NTSC or tuner with or without Radio support

e.g.:

- PAL:
  - TDA5737: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
  - TSA5522: 1.4 GHz I2C-bus controlled synthesizer, I2C 0xc2-0xc3
- NTSC:
  - TDA5731: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
  - TSA5518: no datasheet available on Philips site
- Philips SAA5246 or SAA5284 (or no) Teletext decoder chip with buffer RAM (e.g. Winbond W24257AS-35: 32Kx8 CMOS static RAM) SAA5246 (I2C 0x22) is supported
- 256 bytes EEPROM: Microchip 24LC02B or Philips 8582E2Y with configuration information I2C address 0xa0 (24LC02B also responds to 0xa2-0xaf)
- 1 tuner, 1 composite and (depending on model) 1 S-VHS input
- 14052B: mux for selection of sound source
- sound decoder: TDA9800, MSP34xx (stereo cards)

Askey CPH-Series

Developed by TelSignal(?), OEMed by many vendors (Typhoon, Anubis, Dynalink)

- Card series: - CPH01x: BT848 capture only - CPH03x: BT848 - CPH05x: BT878 with FM - CPH06x: BT878 (w/o FM) - CPH07x: BT878 capture only
- TV standards: - CPH0x0: NTSC-M/M - CPH0x1: PAL-B/G - CPH0x2: PAL-I/I - CPH0x3: PAL-D/K - CPH0x4: SECAM-L/L - CPH0x5: SECAM-B/G - CPH0x6: SECAM-D/K - CPH0x7: PAL-N/N - CPH0x8: PAL-B/H - CPH0x9: PAL-M/M
- CPH03x was often sold as “TV capturer”.

Identifying:

1. 878 cards can be identified by PCI Subsystem-ID: - 144f:3000 = CPH06x - 144F:3002 = CPH05x w/ FM - 144F:3005 = CPH06x_LC (w/o remote control)
2. The cards have a sticker with “CPH”-model on the back.
3. These cards have a number printed on the PCB just above the tuner metal box: - “80-CP2000300-x” = CPH03X - “80-CP2000500-x” = CPH05X - “80-CP2000600-x” = CPH06X / CPH06x_LC

Askey sells these cards as “Magic TView series”, Brand “MagicXpress”. Other OEM often call these “Tview”, “TView99” or else.

Lifeview Flyvideo Series:

The naming of these series differs in time and space.

Identifying:

1. Some models can be identified by PCI subsystem ID:
   - 1852:1852 = Flyvideo 98 FM
   - 1851:1850 = Flyvideo 98
   - 1851:1851 = Flyvideo 98 EZ (capture only)
2. There is a print on the PCB:
   - LR25 = Flyvideo (Zoran ZR36120, SAA7110A)
   - LR26 Rev.N = Flyvideo II (Bt848)
   - LR26 Rev.O = Flyvideo II (Bt878)
   - LR37 Rev.C = Flyvideo EZ (Capture only, ZR36120 + SAA7110)
   - LR38 Rev.A1= Flyvideo II EZ (Bt848 capture only)
   - LR50 Rev.Q = Flyvideo 98 (w/eeprom and PCI subsystem ID)
   - LR50 Rev.W = Flyvideo 98 (no eeprom)
   - LR51 Rev.E = Flyvideo 98 EZ (capture only)
   - LR90 = Flyvideo 2000 (Bt878)
   - LR90 Flyvideo 2000S (Bt878) w/Stereo TV (Package incl. LR91 daughterboard)
   - LR91 = Stereo daughter card for LR90
   - LR97 = Flyvideo DVBS
   - LR99 Rev.E = Low profile card for OEM integration (only internal audio!) bt878
   - LR136 = Flyvideo 2100/3100 (Low profile, SAA7130/SAA7134)
   - LR137 = Flyvideo DV2000/DV3000 (SAA7130/SAA7134 + IEEE1394)
   - LR138 Rev.C= Flyvideo 2000 (SAA7130)
   - LR138 Flyvideo 3000 (SAA7134) w/Stereo TV
     - These exist in variations w/FM and w/Remote sometimes denoted by suffixes “FM” and “R”.
3. You have a laptop (miniPCI card):
   - Product = FlyTV Platinum Mini
   - Model/Chip = LR212/saa7135
   - Lifeview.com.tw states (Feb. 2002): “The FlyVideo2000 and FlyVideo2000s product name have renamed to FlyVideo98.” Their Bt8x8 cards are listed as discontinued.
   - Flyvideo 2000S was probably sold as Flyvideo 3000 in some contries(Europe?). The new Flyvideo 2000/3000 are SAA7130/SAA7134 based.

“Flyvideo II” had been the name for the 848 cards, nowadays (in Germany) this name is re-used for LR50 Rev.W.

The Lifeview website mentioned Flyvideo III at some time, but such a card has not yet been seen (perhaps it was the german name for LR90 [stereo]). These cards are sold by many OEMs too.

FlyVideo A2 (Elta 8680)= LR90 Rev.F (w/Remote, w/o FM, stereo TV by tda9821) {Germany}

Lifeview 3000 (Elta 8681) as sold by Plus(April 2002), Germany = LR138 w/ saa7134

lifeview config coding on gpio pins 0-9

- LR50 rev. Q (“PARTS: 7031505116), Tuner wurde als Nr. 5 erkannt, Eingänge SVideo, TV, Composite, Audio, Remote:
- CP9..1=100001001 (1: 0-Ohm-Widerstand gegen GND unbestückt; 0: bestückt)

Typhoon TV card series:

These can be CPH, Flyvideo, Pixelview or KNC1 series. Typhoon is the brand of Anubis. Model 50680 got re-used, some model no. had different contents over time.

Models:

- 50680 “TV Tuner PCI Pal BG”(old,red package)=can be CPH03x(bt848) or CPH06x(bt878)
- 50680 “TV Tuner Pal BG” (blue package)= Pixelview PV-BT878P+ (Rev 9B)
- 50681 “TV Tuner PCI Pal I” (variant of 50680)
- 50682 “TView TV/FM Tuner Pal BG” = Flyvideo 98FM (LR50 Rev.Q)

Note:

The package has a picture of CPH05x (which would be a real TView)

- 50683 “TV Tuner PCI SECAM” (variant of 50680)
- 50684 “TV Tuner Pal BG” = Pixelview 878TV(Rev.3D)
- 50686 “TV Tuner” = KNC1 TV Station
- 50687 “TV Tuner stereo” = KNC1 TV Station pro
- 50688 “TV Tuner RDS” (black package) = KNC1 TV Station RDS
- 50689 TV SAT DVB-S CARD CI PCI (SAA7146AH, SU1278?) = “KNC1 TV Station DVB-S”
- 50692 “TV/FM Tuner” (small PCB)
- 50694 TV TUNER CARD RDS (PHILIPS CHIPSET SAA7134HL)
- 50696 TV TUNER STEREO (PHILIPS CHIPSET SAA7134HL, MK3ME Tuner)
- 50804 PC-SAT TV/Audio Karte = Techni-PC-Sat (ZORAN 36120PQC, Tuner:Alps)
- 50866 TVIEW SAT RECEIVER+ADR
- 50868 “TV/FM Tuner Pal I” (variant of 50682)
- 50999 “TV/FM Tuner Secam” (variant of 50682)

Guillemot

Models:

- Maxi-TV PCI (ZR36120)
- Maxi TV Video 2 = LR50 Rev.Q (FI1216MF, PAL BG+SECAM)
- Maxi TV Video 3 = CPH064 (PAL BG + SECAM)

Mentor

Mentor TV card (“55-878TV-U1”) = Pixelview 878TV(Rev.3F) (w/FM w/Remote)

Prolink

- TV cards:
  - PixelView Play TV pro - (Model: PV-BT878P+ REV 8E)
  - PixelView Play TV pro - (Model: PV-BT878P+ REV 9D)
  - PixelView Play TV pro - (Model: PV-BT878P+ REV 4C / 8D / 10A)
  - PixelView Play TV - (Model: PV-BT848P+)
  - 878TV - (Model: PV-BT878TV)
- Multimedia TV packages (card + software pack):
  - PixelView Play TV Theater - (Model: PV-M4200) = PixelView Play TV pro + Software
  - PixelView Play TV PAK - (Model: PV-BT878P+ REV 4E)
  - PixelView Play TV/VCR - (Model: PV-M3200 REV 4C / 8D / 10A)
  - PixelView Studio PAK - (Model: M2200 REV 4C / 8D / 10A)
  - PixelView PowerStudio PAK - (Model: PV-M3600 REV 4E)
  - PixelView DigitalVCR PAK - (Model: PV-M2400 REV 4C / 8D / 10A)
  - PixelView PlayTV PAK II (TV/FM card + usb camera) PV-M3800
  - PixelView PlayTV XP PV-M4700,PV-M4700(w/FM)
  - PixelView PlayTV DVR PV-M4600 package contents:PixelView PlayTV pro, windvr & videoMail s/w
- Further Cards:
  - PV-BT878P+rev.9B (Play TV Pro, opt. w/FM w/NICAM)
  - PV-BT878P+rev.2F
  - PV-BT878P Rev.1D (bt878, capture only)
  - XCapture PV-CX881P (cx23881)
  - PlayTV HD PV-CX881PL+, PV-CX881PL+(w/FM) (cx23881)
  - DTV3000 PV-DTV3000P+ DVB-S CI = Twinhan VP-1030
  - DTV2000 DVB-S = Twinhan VP-1020
- Video Conferencing:
  - PixelView Meeting PAK - (Model: PV-BT878P)
  - PixelView Meeting PAK Lite - (Model: PV-BT878P)
  - PixelView Meeting PAK plus - (Model: PV-BT878P+rev 4C/8D/10A)
  - PixelView Capture - (Model: PV-BT848P)
  - PixelView PlayTV USB pro
  - Model No. PV-NT1004+, PV-NT1004+ (w/FM) = NT1004 USB decoder chip + SAA7113 video decoder chip

Dynalink

These are CPH series.

Phoebemicro

- TV Master = CPH030 or CPH060
- TV Master FM = CPH050

Genius/Kye

- Video Wonder/Genius Internet Video Kit = LR37 Rev.C
- Video Wonder Pro II (848 or 878) = LR26

Tekram

- VideoCap C205 (Bt848)
- VideoCap C210 (zr36120 +Philips)
- CaptureTV M200 (ISA)
- CaptureTV M205 (Bt848)

Lucky Star

- Image World Conference TV = LR50 Rev. Q

Leadtek

- WinView 601 (Bt848)
- WinView 610 (Zoran)
- WinFast2000
- WinFast2000 XP

Support for the Leadtek WinView 601 TV/FM

Author of this section: Jon Tombs <jon@gte.esi.us.es>

This card is basically the same as all the rest (Bt484A, Philips tuner), the main difference is that they have attached a programmable attenuator to 3 GPIO lines in order to give some volume control. They have also stuck an infra-red remote control decoded on the board, I will add support for this when I get time (it simple generates an interrupt for each key press, with the key code is placed in the GPIO port).

I don't yet have any application to test the radio support. The tuner frequency setting should work but it is possible that the audio multiplexer is wrong. If it doesn't work, send me email.

- No Thanks to Leadtek they refused to answer any questions about their hardware. The driver was written by visual inspection of the card. If you use this driver, send an email insult to them, and tell them you won't continue buying their hardware unless they support Linux.
- Little thanks to Princeton Technology Corp (http://www.princeton.com.tw) who make the audio attenuator. Their publicly available data-sheet available on their web site doesn't include the chip programming information! Hidden on their server are the full data-sheets, but don't ask how I found it.

To use the driver I use the following options, the tuner and pll settings might be different in your country

insmod videodev insmod i2c scan=1 i2c_debug=0 verbose=0 insmod tuner type=1 debug=0 insmod bttv pll=1 radio=1 card=17

KNC One

- TV-Station
- TV-Station SE (+Software Bundle)
- TV-Station pro (+TV stereo)
- TV-Station FM (+Radio)
- TV-Station RDS (+RDS)
- TV Station SAT (analog satellite)
- TV-Station DVB-S

Note:

newer Cards have saa7134, but model name stayed the same?

Provideo

- **PV951 or PV-951 (also are sold as:** Boeder TV-FM Video Capture Card, Titanmedia Supervision TV-2400, Provideo PV951 TF, 3DeMon PV951, MediaForte TV-Vision PV951, Yoko PV951, Vivanco Tuner Card PCI Art.-Nr.: 68404,) now named PV-951T
- Surveillance Series:
- PV-141
- PV-143
- PV-147
- PV-148 (capture only)
- PV-150
- PV-151
- TV-FM Tuner Series:
- PV-951TDV (tv tuner + 1394)
- PV-951T/TF
- PV-951PT/TF
- PV-956T/TF Low Profile
- PV-911

Highscreen

Models:

- TV Karte = LR50 Rev.S
- TV-Boostar = Terratec Terra TV+ Version 1.0 (Bt848, tda9821) “ceb105.pcb”

Zoltrix

Models:

- Face to Face Capture (Bt848 capture only) (PCB “VP-2848”)
- Face To Face TV MAX (Bt848) (PCB “VP-8482 Rev1.3”)
- Genie TV (Bt878) (PCB “VP-8790 Rev 2.1”)
- Genie Wonder Pro

AVerMedia

- AVer FunTV Lite (ISA, AV3001 chipset) “M101.C”
- AVerTV
- AVerTV Stereo
- AVerTV Studio (w/FM)
- AVerMedia TV98 with Remote
- AVerMedia TV/FM98 Stereo
- AVerMedia TVCAM98
- TVCapture (Bt848)
- TVPhone (Bt848)
- TVCapture98 (=”AVerMedia TV98” in USA) (Bt878)
- TVPhone98 (Bt878, w/FM)

PCB	PCI-ID	Model-Name	Eeprom	Tuner	Sound	Country
M101.C	ISA !					
M108-B	Bt848		–	FR1236		US [2], [3]
M1A8-A	Bt848	AVer TV-Phone		FM1216	–	
M168-T	1461:0003	AVerTV Studio	48:17	FM1216	TDA9840T	D [1] w/FM w/Remote
M168-U	1461:0004	TVCapture98	40:11	FI1216	–	D w/Remote
M168II-B	1461:0003	Medion MD9592	48:16	FM1216	TDA9873H	D w/FM

- US site has different drivers for (as of 09/2002):
  - EZ Capture/InterCam PCI (BT-848 chip)
  - EZ Capture/InterCam PCI (BT-878 chip)
  - TV-Phone (BT-848 chip)
  - TV98 (BT-848 chip)
  - TV98 With Remote (BT-848 chip)
  - TV98 (BT-878 chip)
  - TV98 With Remote (BT-878)
  - TV/FM98 (BT-878 chip)
  - AVerTV
  - AverTV Stereo
  - AVerTV Studio

[2] Sony NE41S soldered (stereo sound?)
[3] Daughterboard M118-A w/ pic 16c54 and 4 MHz quartz
[1] Daughterboard MB68-A with TDA9820T and TDA9840T

DE hat diverse Treiber fuer diese Modelle (Stand 09/2002):

- TVPhone (848) mit Philips tuner FR12X6 (w/ FM radio)
- TVPhone (848) mit Philips tuner FM12X6 (w/ FM radio)
- TVCapture (848) w/Philips tuner FI12X6
- TVCapture (848) non-Philips tuner
- TVCapture98 (Bt878)
- TVPhone98 (Bt878)
- AVerTV und TVCapture98 w/VCR (Bt 878)
- AVerTVStudio und TVPhone98 w/VCR (Bt878)
- AVerTV GO Serie (Kein SVideo Input)
- AVerTV98 (BT-878 chip)
- AVerTV98 mit Fernbedienung (BT-878 chip)
- AVerTV/FM98 (BT-878 chip)
- VDOmate (www.averm.com.cn) = M168U ?

Aimslab

Models:

- Video Highway or “Video Highway TR200” (ISA)
- Video Highway Xtreme (aka “VHX”) (Bt848, FM w/ TEA5757)

IXMicro (former: IMS=Integrated Micro Solutions)

Models:

- IXTV BT848 (=TurboTV)
- IXTV BT878
- IMS TurboTV (Bt848)

Lifetec/Medion/Tevion/Aldi

Models:

- LT9306/MD9306 = CPH061
- LT9415/MD9415 = LR90 Rev.F or Rev.G
- MD9592 = Avermedia TVphone98 (PCI_ID=1461:0003), PCB-Rev=M168II-B (w/TDA9873H)
- MD9717 = KNC One (Rev D4, saa7134, FM1216 MK2 tuner)
- MD5044 = KNC One (Rev D4, saa7134, FM1216ME MK3 tuner)

Modular Technologies (www.modulartech.com) UK

Models:

- MM100 PCTV (Bt848)
- MM201 PCTV (Bt878, Bt832) w/ Quartzsight camera

- MM202 PCTV (Bt878, Bt832, tda9874)
- MM205 PCTV (Bt878)
- MM210 PCTV (Bt878) (Galaxy TV, Galaxymedia ?)

Terratec

Models:

- Terra TV+ Version 1.0 (Bt848), “ceb105.PCB” printed on the PCB, TDA9821
- Terra TV+ Version 1.1 (Bt878), “LR74 Rev.E” printed on the PCB, TDA9821
- Terra TValueRadio, “LR102 Rev.C” printed on the PCB
- Terra TV/Radio+ Version 1.0, “80-CP2830100-0” TTTV3 printed on the PCB, “CPH010-E83” on the back, SAA6588T, TDA9873H
- Terra TValue Version BT878, “80-CP2830110-0 TTTV4” printed on the PCB, “CPH011-D83” on back
- Terra TValue Version 1.0 “ceb105.PCB” (really identical to Terra TV+ Version 1.0)
- Terra TValue New Revision “LR102 Rec.C”
- Terra Active Radio Upgrade (tea5757h, saa6588t)
- LR74 is a newer PCB revision of ceb105 (both incl. connector for Active Radio Upgrade)
- Cinergy 400 (saa7134), “E877 11(S)”, “PM820092D” printed on PCB
- Cinergy 600 (saa7134)

Technisat

Models:

- Discos ADR PC-Karte ISA (no TV!)
- Discos ADR PC-Karte PCI (probably no TV?)
- Techni-PC-Sat (Sat. analog) Rev 1.2 (zr36120, vpx3220, stv0030, saa5246, BSJE3-494A)
- Mediafocus I (zr36120/zr36125, drp3510, Sat. analog + ADR Radio)
- Mediafocus II (saa7146, Sat. analog)
- SatADR Rev 2.1 (saa7146a, saa7113h, stv0056a, msp3400c, drp3510a, BSKE3-307A)
- SkyStar 1 DVB (AV7110) = Technotrend Premium
- SkyStar 2 DVB (B2C2) (=Sky2PC)

Siemens

Multimedia eXtension Board (MXB) (SAA7146, SAA7111)

Powercolor

Models:

- **MTV878** Package comes with different contents:
    1. pcb “MTV878” (CARD=75)
    2. Pixelview Rev. 4_
- MTV878R w/Remote Control

- MTV878F w/Remote Control w/FM radio

Pinnacle

PCTV models:

- Mirovideo PCTV (Bt848)
- Mirovideo PCTV SE (Bt848)
- Mirovideo PCTV Pro (Bt848 + Daughterboard for TV Stereo and FM)
- Studio PCTV Rave (Bt848 Version = Mirovideo PCTV)
- Studio PCTV Rave (Bt878 package w/o infrared)
- Studio PCTV (Bt878)
- Studio PCTV Pro (Bt878 stereo w/ FM)
- Pinnacle PCTV (Bt878, MT2032)
- Pinnacle PCTV Pro (Bt878, MT2032)
- Pinncale PCTV Sat (bt878a, HM1821/1221) [”Conexant CX24110 with CX24108 tuner, aka HM1221/HM1811”]
- Pinnacle PCTV Sat XE

M(J)PEG capture and playback models:

- DC1+ (ISA)
- DC10 (zr36057, zr36060, saa7110, adv7176)
- DC10+ (zr36067, zr36060, saa7110, adv7176)
- DC20 (ql16x24b,zr36050, zr36016, saa7110, saa7187 ...)
- DC30 (zr36057, zr36050, zr36016, vpx3220, adv7176, ad1843, tea6415, miro FST97A1)
- DC30+ (zr36067, zr36050, zr36016, vpx3220, adv7176)
- DC50 (zr36067, zr36050, zr36016, saa7112, adv7176 (2 pcs.?), ad1843, miro FST97A1, Lattice ???)

Lenco

Models:

- MXR-9565 (=Technisat Mediafocus?)
- MXR-9571 (Bt848) (=CPH031?)
- MXR-9575
- MXR-9577 (Bt878) (=Prolink 878TV Rev.3x)
- MXTV-9578CP (Bt878) (= Prolink PV-BT878P+4E)

Iomega

Buz (zr36067, zr36060, saa7111, saa7185)

LML

LML33 (zr36067, zr36060, bt819, bt856)

Grandtec

Models:

- Grand Video Capture (Bt848)
- Multi Capture Card (Bt878)

Koutech

Models:

- KW-606 (Bt848)
- KW-607 (Bt848 capture only)
- KW-606RSF
- KW-607A (capture only)
- KW-608 (Zoran capture only)

IODATA (jp)

Models:

- GV-BCTV/PCI
- GV-BCTV2/PCI
- GV-BCTV3/PCI
- GV-BCTV4/PCI
- GV-VCP/PCI (capture only)
- GV-VCP2/PCI (capture only)

Canopus (jp)

WinDVR = Kworld “KW-TVL878RF”

www.sigmacom.co.kr

Sigma Cyber TV II

www.sasem.co.kr

Litte OnAir TV

hama

TV/Radio-Tuner Card, PCI (Model 44677) = CPH051

Sigma Designs

Hollywood plus (em8300, em9010, adv7175), (PCB “M340-10”) MPEG DVD decoder

Formac

Models:

- iProTV (Card for iMac Mezzanine slot, Bt848+SCSI)
- ProTV (Bt848)
- ProTV II = ProTV Stereo (Bt878) [”stereo” means FM stereo, tv is still mono]

ATI

Models:

- TV-Wonder
- TV-Wonder VE

Diamond Multimedia

DTV2000 (Bt848, tda9875)

Aopen

- VA1000 Plus (w/ Stereo)
- VA1000 Lite
- VA1000 (=LR90)

Intel

Models:

- Smart Video Recorder (ISA full-length)
- Smart Video Recorder pro (ISA half-length)
- Smart Video Recorder III (Bt848)

STB

Models:

- STB Gateway 6000704 (bt878)
- STB Gateway 6000699 (bt848)
- STB Gateway 6000402 (bt848)
- STB TV130 PCI

Videologic

Models:

- Captivator Pro/TV (ISA?)
- Captivator PCI/VC (Bt848 bundled with camera) (capture only)

Technotrend

Models:

- TT-SAT PCI (PCB “Sat-PCI Rev.:1.3.1”; zr36125, vpx3225d, stc0056a, Tuner:BSKE6-155A
- **TT-DVB-Sat**
  - revisions 1.1, 1.3, 1.5, 1.6 and 2.1
  - This card is sold as OEM from:
    * Siemens DVB-s Card
    * Hauppauge WinTV DVB-S
    * Technisat SkyStar 1 DVB
    * Galaxis DVB Sat
  - Now this card is called TT-PCline Premium Family
  - TT-Budget (saa7146, bsru6-701a) This card is sold as OEM from:
    * Hauppauge WinTV Nova
    * Satelco Standard PCI (DVB-S)
  - TT-DVB-C PCI

Teles

DVB-s (Rev. 2.2, BSRV2-301A, data only?)

Remote Vision

MX RV605 (Bt848 capture only)

Boeder

Models:

- PC ChatCam (Model 68252) (Bt848 capture only)
- Tv/Fm Capture Card (Model 68404) = PV951

Media-Surfer (esc-kathrein.de)

Models:

- Sat-Surfer (ISA)
- Sat-Surfer PCI = Techni-PC-Sat
- Cable-Surfer 1
- Cable-Surfer 2
- Cable-Surfer PCI (zr36120)
- Audio-Surfer (ISA Radio card)

Jetway (www.jetway.com.tw)

Models:

- JW-TV 878M
- JW-TV 878 = KWorld KW-TV878RF

Galaxis

Models:

- Galaxis DVB Card S CI
- Galaxis DVB Card C CI
- Galaxis DVB Card S
- Galaxis DVB Card C
- Galaxis plug.in S [neuer Name: Galaxis DVB Card S CI

Hauppauge

Models:

- many many WinTV models ...
- WinTV DVBs = Technotrend Premium 1.3
- WinTV NOVA = Technotrend Budget 1.1 “S-DVB DATA”
- WinTV NOVA-CI “SDVBACI”
- WinTV Nova USB (=Technotrend USB 1.0)
- WinTV-Nexus-s (=Technotrend Premium 2.1 or 2.2)
- WinTV PVR
- WinTV PVR 250
- WinTV PVR 450

US models

-990 WinTV-PVR-350 (249USD) (iTVC15 chipset + radio) -980 WinTV-PVR-250 (149USD) (iTVC15 chipset) -880 WinTV-PVR-PCI (199USD) (KFIR chipset + bt878) -881 WinTV-PVR-USB -190 WinTV-GO -191 WinTV-GO-FM -404 WinTV -401 WinTV-radio -495 WinTV-Theater -602 WinTV-USB -621 WinTV-USB-FM -600 USB-Live -698 WinTV-HD -697 WinTV-D -564 WinTV-Nexus-S

Deutsche Modelle:

-603 WinTV GO -719 WinTV Primio-FM -718 WinTV PCI-FM -497 WinTV Theater -569 WinTV USB -568 WinTV USB-FM -882 WinTV PVR -981 WinTV PVR 250 -891 WinTV-PVR-USB -541 WinTV Nova -488 WinTV Nova-Ci -564 WinTV-Nexus-s -727 WinTV-DVB-c -545 Common Interface -898 WinTV-Nova-USB

UK models:

-607 WinTV Go -693,793 WinTV Primio FM -647,747 WinTV PCI FM -498 WinTV Theater -883 WinTV PVR -893 WinTV PVR USB (Duplicate entry) -566 WinTV USB (UK) -573 WinTV USB FM -429 Impact VCB (bt848) -600 USB Live (Video-In 1x Comp, 1xSVHS) -542 WinTV Nova -717 WinTV DVB-S -909 Nova-t PCI -893 Nova-t USB (Duplicate entry) -802 MyTV -804 MyView -809 MyVideo -872 MyTV2Go FM -546 WinTV Nova-S CI -543 WinTV Nova -907 Nova-S USB -908 Nova-T USB -717 WinTV Nexus-S -157 DEC3000-s Standalone + USB

Spain:

-685 WinTV-Go -690 WinTV-PrimioFM -416 WinTV-PCI Nicam Estereo -677 WinTV-PCI-FM -699 WinTV-Theater -683 WinTV-USB -678 WinTV-USB-FM -983 WinTV-PVR-250 -883 WinTV-PVR-PCI -993 WinTV-PVR-350 -893 WinTV-PVR-USB -728 WinTV-DVB-C PCI -832 MyTV2Go -869 MyTV2Go-FM -805 MyVideo (USB)

Matrix-Vision

Models:

- MATRIX-Vision MV-Delta
- MATRIX-Vision MV-Delta 2
- MVsigma-SLC (Bt848)

Conceptronic (.net)

Models:

- TVCON FM, TV card w/ FM = CPH05x
- TVCON = CPH06x

BestData

Models:

- HCC100 = VCC100rev1 + camera
- VCC100 rev1 (bt848)
- VCC100 rev2 (bt878)

Gallant (www.gallantcom.com) www.minton.com.tw

Models:

- Intervision IV-510 (capture only bt8x8)
- Intervision IV-550 (bt8x8)
- Intervision IV-100 (zoran)
- Intervision IV-1000 (bt8x8)

Asonic (www.asonic.com.cn) (website down)

SkyEye tv 878

Hoontech

878TV/FM

Teppro (www.itcteppro.com.tw)

Models:

- ITC PCITV (Card Ver 1.0) “Teppro TV1/TVFM1 Card”
- ITC PCITV (Card Ver 2.0)
- ITC PCITV (Card Ver 3.0) = “PV-BT878P+ (REV.9D)”

- ITC PCITV (Card Ver 4.0)
- TEPPRO IV-550 (For BT848 Main Chip)
- ITC DSTTV (bt878, satellite)
- ITC VideoMaker (saa7146, StreamMachine sm2110, tvtuner) “PV-SM2210P+ (REV:1C)”

Kworld (www.kworld.com.tw)

PC TV Station:

- KWORLD KW-TV878R TV (no radio)
- KWORLD KW-TV878RF TV (w/ radio)
- KWORLD KW-TVL878RF (low profile)
- KWORLD KW-TV713XRF (saa7134)
  MPEG TV Station (same cards as above plus WinDVR Software MPEG en/decoder)
- KWORLD KW-TV878R -Pro TV (no Radio)
- KWORLD KW-TV878RF-Pro TV (w/ Radio)
- KWORLD KW-TV878R -Ultra TV (no Radio)
- KWORLD KW-TV878RF-Ultra TV (w/ Radio)

JTT/ Justy Corp.(http://www.jtt.ne.jp/)

JTT-02 (JTT TV) “TV watchmate pro” (bt848)

ADS www.adstech.com

Models:

- Channel Surfer TV (CHX-950)
- Channel Surfer TV+FM (CHX-960FM)

AVEC www.prochips.com

AVEC Intercapture (bt848, tea6320)

NoBrand

TV Excel = Australian Name for “PV-BT878P+ 8E” or “878TV Rev.3_”

Mach www.machspeed.com

Mach TV 878

Eline www.eline-net.com/

Models:

- Eline Vision TVMaster / TVMaster FM (ELV-TVM/ ELV-TVM-FM) = LR26 (bt878)
- Eline Vision TVMaster-2000 (ELV-TVM-2000, ELV-TVM-2000-FM)= LR138 (saa713x)

Spirit

- Spirit TV Tuner/Video Capture Card (bt848)

Boser www.boser.com.tw

Models:

- HS-878 Mini PCI Capture Add-on Card
- HS-879 Mini PCI 3D Audio and Capture Add-on Card (w/ ES1938 Solo-1)

Satelco www.citycom-gmbh.de, www.satelco.de

Models:

- TV-FM =KNC1 saa7134
- Standard PCI (DVB-S) = Technotrend Budget
- Standard PCI (DVB-S) w/ CI
- Satelco Highend PCI (DVB-S) = Technotrend Premium

Sensoray www.sensoray.com

Models:

- Sensoray 311 (PC/104 bus)
- Sensoray 611 (PCI)

CEI (Chartered Electronics Industries Pte Ltd [CEI] [FCC ID HBY])

Models:

- TV Tuner - HBY-33A-RAFFLES Brooktree Bt848KPF + Philips
- TV Tuner MG9910 - HBY33A-TVO CEI + Philips SAA7110 + OKI M548262 + ST STV8438CV
- Primetime TV (ISA)
  - acquired by Singapore Technologies
  - now operating as Chartered Semiconductor Manufacturing
  - Manufacturer of video cards is listed as:
    * Cogent Electronics Industries [CEI]

AITech

Models:

- Wavewatcher TV (ISA)
- AITech WaveWatcher TV-PCI = can be LR26 (Bt848) or LR50 (BT878)
- WaveWatcher TVR-202 TV/FM Radio Card (ISA)

MAXRON

Maxron MaxTV/FM Radio (KW-TV878-FNT) = Kworld or JW-TV878-FBK

www.ids-imaging.de

Models:

- Falcon Series (capture only)

In USA: http://www.theimagingsource.com/ - DFG/LC1

www.sknet-web.co.jp

SKnet Monster TV (saa7134)

A-Max www.amaxhk.com (Colormax, Amax, Napa)

APAC Viewcomp 878

Cybertainment

Models:

- CyberMail AV Video Email Kit w/ PCI Capture Card (capture only)
- CyberMail Xtreme

These are Flyvideo

VCR (http://www.vcrinc.com/)

Video Catcher 16

Twinhan

Models:

- DST Card/DST-IP (bt878, twinhan asic) VP-1020 - Sold as:
  - KWorld DVBS Satellite TV-Card
  - Powercolor DSTV Satellite Tuner Card
  - Prolink Pixelview DTV2000
  - Provideo PV-911 Digital Satellite TV Tuner Card With Common Interface ?
- DST-CI Card (DVB Satellite) VP-1030
- DCT Card (DVB cable)

MSI

Models:

- MSI TV@nywhere Tuner Card (MS-8876) (CX23881/883) Not Bt878 compatible.
- MS-8401 DVB-S

Focus www.focusinfo.com

InVideo PCI (bt878)

Sdisilk www.sdisilk.com/

Models:

- SDI Silk 100
- SDI Silk 200 SDI Input Card

www.euresys.com

PICOLO series

PMC/Pace

www.pacecom.co.uk website closed

Mercury www.kobian.com (UK and FR)

Models:

- LR50
- LR138RBG-Rx == LR138

TEC sound

TV-Mate = Zoltrix VP-8482

Though educated googling found: www.techmakers.com

(package and manuals don't have any other manufacturer info) TecSound

Lorenzen www.lorenzen.de

SL DVB-S PCI = Technotrend Budget PCI (su1278 or bsru version)

Origo (.uk) www.origo2000.com

PC TV Card = LR50

I/O Magic www.iomagic.com

PC PVR - Desktop TV Personal Video Recorder DR-PCTV100 = Pinnacle ROB2D-51009464 4.0 + Cyberlink PowerVCR II

Arowana

TV-Karte / Poso Power TV (?) = Zoltrix VP-8482 (?)

iTVC15 boards

kuroutoshikou.com ITVC15 yuan.com MPG160 PCI TV (Internal PCI MPEG2 encoder card plus TV-tuner)

Asus www.asuscom.com

Models:

- Asus TV Tuner Card 880 NTSC (low profile, cx23880)
- Asus TV (saa7134)

Hoontech

http://www.hoontech.de/

- HART Vision 848 (H-ART Vision 848)
- HART Vision 878 (H-Art Vision 878)

4.7.10 Chips used at bttv devices

- all boards:
  - Brooktree Bt848/848A/849/878/879: video capture chip
- Board specific
  - Miro PCTV:
    - Philips or Temic Tuner
  - Hauppauge Win/TV pci (version 405):
    - Microchip 24LC02B or Philips 8582E2Y:
      - 256 Byte EEPROM with configuration information
      - I2C 0xa0-0xa1, (24LC02B also responds to 0xa2-0xaf)
    - Philips SAA5246AGP/E: Videotext decoder chip, I2C 0x22-0x23
    - TDA9800: sound decoder
    - Winbond W24257AS-35: 32Kx8 CMOS static RAM (Videotext buffer mem)
    - 14052B: analog switch for selection of sound source
- PAL:
  - TDA5737: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
  - TSA5522: 1.4 GHz I2C-bus controlled synthesizer, I2C 0xc2-0xc3
- NTSC:
  - TDA5731: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
  - TSA5518: no datasheet available on Philips site
- STB TV pci:
  - ???
  - if you want better support for STB cards send me info! Look at the board! What chips are on it?

4.7.11 Specs

Philips http://www.Semiconductors.COM/pip/

Conexant http://www.conexant.com/

Micronas http://www.micronas.com/en/home/index.html

4.7.12 Thanks

Many thanks to:

- Markus Schroeder <schroedm@uni-duesseldorf.de> for information on the Bt848 and tuner programming and his control program xtvc.
- Martin Buck <martin-2.buck@student.uni-ulm.de> for his great Videotext package.
- Gerd Hoffmann for the MSP3400 support and the modular I2C, tuner, ... support.
- MATRIX Vision for giving us 2 cards for free, which made support of single crystal operation possible.
- MIRO for providing a free PCTV card and detailed information about the components on their cards. (E.g. how the tuner type is detected) Without their card I could not have debugged the NTSC mode.
- Hauppauge for telling how the sound input is selected and what components they do and will use on their radio cards. Also many thanks for faxing me the FM1216 data sheet.

4.7.13 Contributors

Michael Chu <mmchu@pobox.com> AverMedia fix and more flexible card recognition

Alan Cox <alan@lxorguk.ukuu.org.uk> Video4Linux interface and 2.1.x kernel adaptation

Chris Kleitsch Hardware I2C

Gerd Hoffmann Radio card (ITT sound processor)

bigfoot <bigfoot@net-way.net>

Ragnar Hojland Espinosa <ragnar@macula.net> ConferenceTV card

- **many more (please mail me if you are missing in this list and would** like to be mentioned)

4.8 The cafe_ccic driver

Author: Jonathan Corbet <corbet@lwn.net>

4.8.1 Introduction

"cafe_ccic" is a driver for the Marvell 88ALP01 "cafe" CMOS camera controller. This is the controller found in first-generation OLPC systems, and this driver was written with support from the OLPC project.

Current status: the core driver works. It can generate data in YUV422, RGB565, and RGB444 formats. (Anybody looking at the code will see RGB32 as well, but that is a debugging aid which will be removed shortly). VGA and QVGA modes work; CIF is there but the colors remain funky. Only the OV7670 sensor is known to work with this controller at this time.

To try it out: either of these commands will work:

```
$ mplayer tv:// -tv driver=v4l2:width=640:height=480 -nosound
$ mplayer tv:// -tv driver=v4l2:width=640:height=480:outfmt=bgr16 -nosound
```

The "xawtv" utility also works; gqcam does not, for unknown reasons.

4.8.2 Load time options

There are a few load-time options, most of which can be changed after loading via sysfs as well:

- alloc_bufs_at_load: Normally, the driver will not allocate any DMA buffers until the time comes to transfer data. If this option is set, then worst-case-sized buffers will be allocated at module load time. This option nails down the memory for the life of the module, but perhaps decreases the chances of an allocation failure later on.
- dma_buf_size: The size of DMA buffers to allocate. Note that this option is only consulted for load-time allocation; when buffers are allocated at run time, they will be sized appropriately for the current camera settings.
- n_dma_bufs: The controller can cycle through either two or three DMA buffers. Normally, the driver tries to use three buffers; on faster systems, however, it will work well with only two.
- min_buffers: The minimum number of streaming I/O buffers that the driver will consent to work with. Default is one, but, on slower systems, better behavior with mplayer can be achieved by setting to a higher value (like six).
- max_buffers: The maximum number of streaming I/O buffers; default is ten. That number was carefully picked out of a hat and should not be assumed to actually mean much of anything.
- flip: If this boolean parameter is set, the sensor will be instructed to invert the video image. Whether it makes sense is determined by how your particular camera is mounted.

4.9 The cpia2 driver

Authors: Peter Pregler <Peter_Pregler@email.com>, Scott J. Bertin <scottbertin@yahoo.com>, and Jarl Totland <Jarl.Totland@bdc.no> for the original cpia driver, which this one was modelled from.

4.9.1 Introduction

This is a driver for STMicroelectronics's CPiA2 (second generation Colour Processor Interface ASIC) based cameras. This camera outputs an MJPEG stream at up to vga size. It implements the Video4Linux interface as much as possible. Since the V4L interface does not support compressed formats, only an mjpeg enabled application can be used with the camera. We have modified the gqcam application to view this stream.

The driver is implemented as two kernel modules. The cpia2 module contains the camera functions and the V4L interface. The cpia2_usb module contains usb specific functions. The main reason for this was the size of the module was getting out of hand, so I separated them. It is not likely that there will be a parallel port version.

4.9.2 Features

- Supports cameras with the Vision stv6410 (CIF) and stv6500 (VGA) cmos sensors. I only have the vga sensor, so can't test the other.
- Image formats: VGA, QVGA, CIF, QCIF, and a number of sizes in between. VGA and QVGA are the native image sizes for the VGA camera. CIF is done in the coprocessor by scaling QVGA. All other sizes are done by clipping.
- Palette: YCrCb, compressed with MJPEG.
- Some compression parameters are settable.
- Sensor framerate is adjustable (up to 30 fps CIF, 15 fps VGA).
- Adjust brightness, color, contrast while streaming.
- Flicker control settable for 50 or 60 Hz mains frequency.

4.9.3 Making and installing the stv672 driver modules

Requirements

Video4Linux must be either compiled into the kernel or available as a module. Video4Linux2 is automatically detected and made available at compile time.

Setup

Use 'modprobe cpia2' to load and 'modprobe -r cpia2' to unload. This may be done automatically by your distribution.

Driver options

Option	Description
video_nr	video device to register (0=/dev/video0, etc) range -1 to 64. default is -1 (first available) If you have more than 1 camera, this MUST be -1.
buffer_size	Size for each frame buffer in bytes (default 68k)
num_buffers	Number of frame buffers (1-32, default 3)
alternate	USB Alternate (2-7, default 7)
flicker_freq	Frequency for flicker reduction(50 or 60, default 60)
flicker_mode	0 to disable, or 1 to enable flicker reduction. (default 0). This is only effective if the camera uses a stv0672 coprocessor.

Setting the options

If you are using modules, edit /etc/modules.conf and add an options line like this:

```
options cpia2 num_buffers=3 buffer_size=65535
```

If the driver is compiled into the kernel, at boot time specify them like this:

```
cpia2.num_buffers=3 cpia2.buffer_size=65535
```

What buffer size should I use?

The maximum image size depends on the alternate you choose, and the frame rate achieved by the camera. If the compression engine is able to keep up with the frame rate, the maximum image size is given by the table below.

The compression engine starts out at maximum compression, and will increase image quality until it is close to the size in the table. As long as the compression engine can keep up with the frame rate, after a short time the images will all be about the size in the table, regardless of resolution.

At low alternate settings, the compression engine may not be able to compress the image enough and will reduce the frame rate by producing larger images.

The default of 68k should be good for most users. This will handle any alternate at frame rates down to 15fps. For lower frame rates, it may be necessary to increase the buffer size to avoid having frames dropped due to insufficient space.

Alternate	bytes/ms	15fps	30fps
2	128	8533	4267
3	384	25600	12800
4	640	42667	21333
5	768	51200	25600
6	896	59733	29867
7	1023	68200	34100

Table: Image size(bytes)

How many buffers should I use?

For normal streaming, 3 should give the best results. With only 2, it is possible for the camera to finish sending one image just after a program has started reading the other. If this happens, the driver must drop a frame. The exception to this is if you have a heavily loaded machine. In this case use 2 buffers. You are probably not reading at the full frame rate. If the camera can send multiple images before a read finishes, it could overwrite the third buffer before the read finishes, leading to a corrupt image. Single and double buffering have extra checks to avoid overwriting.

Using the camera

We are providing a modified gqcam application to view the output. In order to avoid confusion, here it is called mview. There is also the qx5view program which can also control the lights on the qx5 microscope. MJPEG Tools (http://mjpeg.sourceforge.net) can also be used to record from the camera.

Notes to developers

- This is a driver version stripped of the 2.4 back compatibility and old MJPEG ioctl API. See cpia2.sf.net for 2.4 support.

Programmer's overview of cpia2 driver

Cpia2 is the second generation video coprocessor from VLSI Vision Ltd (now a division of ST Microelectronics). There are two versions. The first is the STV0672, which is capable of up to 30 frames per second (fps) in frame sizes up to CIF, and 15 fps for VGA frames. The STV0676 is an improved version, which can handle up to 30 fps VGA. Both coprocessors can be attached to two CMOS sensors - the vvl6410 CIF sensor and the vvl6500 VGA sensor. These will be referred to as the 410 and the 500 sensors, or the CIF and VGA sensors.

The two chipsets operate almost identically. The core is an 8051 processor, running two different versions of firmware. The 672 runs the VP4 video processor code, the 676 runs VP5. There are a few differences in register mappings for the two chips. In these cases, the symbols defined in the header files are marked with VP4 or VP5 as part of the symbol name.

The cameras appear externally as three sets of registers. Setting register values is the only way to control the camera. Some settings are interdependant, such as the sequence required to power up the camera. I will try to make note of all of these cases.

The register sets are called blocks. Block 0 is the system block. This section is always powered on when the camera is plugged in. It contains registers that control housekeeping functions such as powering up the video processor. The video processor is the VP block. These registers control how the video from the sensor is processed. Examples are timing registers, user mode (vga, qvga), scaling, cropping, framerates, and so on. The last block is the video compressor (VC). The video stream sent from the camera is compressed as Motion JPEG (JPEGA). The VC controls all of the compression parameters. Looking at the file cpia2_registers.h, you can get a full view of these registers and the possible values for most of them.

One or more registers can be set or read by sending a usb control message to the camera. There are three modes for this. Block mode requests a number of contiguous registers. Random mode reads or writes

random registers with a tuple structure containing address/value pairs. The repeat mode is only used by VP4 to load a firmware patch. It contains a starting address and a sequence of bytes to be written into a gpio port.

4.10 The cx18 driver

Note:

This documentation is outdated.

Some notes regarding the cx18 driver for the Conexant CX23418 MPEG encoder chip:

1. Currently supported are:
   - Hauppauge HVR-1600
   - Compro VideoMate H900
   - Yuan MPC718
   - Conexant Raptor PAL/SECAM devkit
2. Some people have problems getting the i2c bus to work. The symptom is that the eeprom cannot be read and the card is unusable. This is probably fixed, but if you have problems then post to the video4linux or ivtv-users mailing list.
3. VBI (raw or sliced) has not yet been implemented.
4. MPEG indexing is not yet implemented.
5. The driver is still a bit rough around the edges, this should improve over time.

Firmware:

You can obtain the firmware files here:

http://dl.ivtvdriver.org/ivtv/firmware/cx18-firmware.tar.gz

Untar and copy the .fw files to your firmware directory.

4.11 The cx2341x driver

4.11.1 Memory at cx2341x chips

This section describes the cx2341x memory map and documents some of the register space.

Note:

the memory long words are little-endian (‘intel format’).

Warning:

This information was figured out from searching through the memory and registers, this information may not be correct and is certainly not complete, and was not derived from anything more than searching through the memory space with commands like:

```
ivtvctl -O min=0x02000000,max=0x020000ff
```

So take this as is, I'm always searching for more stuff, it's a large register space :-).

Memory Map

The cx2341x exposes its entire 64M memory space to the PCI host via the PCI BAR0 (Base Address Register 0). The addresses here are offsets relative to the address held in BAR0.

```
0x00000000-0x00ffffff Encoder memory space
0x00000000-0x0003ffff Encode.rom
???-???         MPEG buffer(s)
???-???         Raw video capture buffer(s)
???-???         Raw audio capture buffer(s)
???-???         Display buffers (6 or 9)

0x01000000-0x01ffffff Decoder memory space
0x01000000-0x0103ffff Decode.rom
???-???         MPEG buffers(s)
0x0114b000-0x0115afff Audio.rom (deprecated?)

0x02000000-0x0200ffff Register Space
```

Registers

The registers occupy the 64k space starting at the 0x02000000 offset from BAR0. All of these registers are 32 bits wide.

```
DMA Registers 0x000-0xff:

0x00 - Control:
        0=reset/cancel, 1=read, 2=write, 4=stop
0x04 - DMA status:
        1=read busy, 2=write busy, 4=read error, 8=write error, 16=link list error
0x08 - pci DMA pointer for read link list
0x0c - pci DMA pointer for write link list
0x10 - read/write DMA enable:
        1=read enable, 2=write enable
0x14 - always 0xffffffff, if set any lower instability occurs, 0x00 crashes
0x18 - ??
0x1c - always 0x20 or 32, smaller values slow down DMA transactions
0x20 - always value of 0x780a010a
0x24-0x3c - usually just random values???
0x40 - Interrupt status
0x44 - Write a bit here and shows up in Interrupt status 0x40
0x48 - Interrupt Mask
0x4C - always value of 0xfffdffff,
        if changed to 0xffffffff DMA write interrupts break.
0x50 - always 0xffffffff
0x54 - always 0xffffffff (0x4c, 0x50, 0x54 seem like interrupt masks, are
        3 processors on chip, Java ones, VPU, SPU, APU, maybe these are the
        interrupt masks???).
```

```
0x60-0x7C - random values
0x80 - first write linked list reg, for Encoder Memory addr
0x84 - first write linked list reg, for pci memory addr
0x88 - first write linked list reg, for length of buffer in memory addr
       (|0x80000000 or this for last link)
0x8c-0xdc - rest of write linked list reg, 8 sets of 3 total, DMA goes here
       from linked list addr in reg 0x0c, firmware must push through or
       something.
0xe0 - first (and only) read linked list reg, for pci memory addr
0xe4 - first (and only) read linked list reg, for Decoder memory addr
0xe8 - first (and only) read linked list reg, for length of buffer
0xec-0xff - Nothing seems to be in these registers, 0xec-f4 are 0x00000000.
```

Memory locations for Encoder Buffers 0x700-0x7ff:

These registers show offsets of memory locations pertaining to each buffer area used for encoding, have to shift them by <<1 first.

- 0x07F8: Encoder SDRAM refresh
- 0x07FC: Encoder SDRAM pre-charge

Memory locations for Decoder Buffers 0x800-0x8ff:

These registers show offsets of memory locations pertaining to each buffer area used for decoding, have to shift them by <<1 first.

- 0x08F8: Decoder SDRAM refresh
- 0x08FC: Decoder SDRAM pre-charge

Other memory locations:

- 0x2800: Video Display Module control
- 0x2D00: AO (audio output?) control
- 0x2D24: Bytes Flushed
- 0x7000: LSB I2C write clock bit (inverted)
- 0x7004: LSB I2C write data bit (inverted)
- 0x7008: LSB I2C read clock bit
- 0x700c: LSB I2C read data bit
- 0x9008: GPIO get input state
- 0x900c: GPIO set output state
- 0x9020: GPIO direction (Bit7 (GPIO 0..7) - 0:input, 1:output)
- 0x9050: SPU control
- 0x9054: Reset HW blocks
- 0x9058: VPU control
- 0xA018: Bit6: interrupt pending?
- 0xA064: APU command

Interrupt Status Register

The definition of the bits in the interrupt status register 0x0040, and the interrupt mask 0x0048. If a bit is cleared in the mask, then we want our ISR to execute.

- bit 31 Encoder Start Capture

- bit 30 Encoder EOS
- bit 29 Encoder VBI capture
- bit 28 Encoder Video Input Module reset event
- bit 27 Encoder DMA complete
- bit 24 Decoder audio mode change detection event (through event notification)
- bit 22 Decoder data request
- bit 20 Decoder DMA complete
- bit 19 Decoder VBI re-insertion
- bit 18 Decoder DMA err (linked-list bad)

4.11.2 Missing documentation

- Encoder API post(?)
- Decoder API post(?)
- Decoder VTRACE event

4.11.3 The cx2341x firmware upload

This document describes how to upload the cx2341x firmware to the card.

How to find

See the web pages of the various projects that uses this chip for information on how to obtain the firmware. The firmware stored in a Windows driver can be detected as follows:

- Each firmware image is 256k bytes.
- The 1st 32-bit word of the Encoder image is 0x0000da7
- The 1st 32-bit word of the Decoder image is 0x00003a7
- The 2nd 32-bit word of both images is 0xaa55bb66

How to load

- Issue the FWapi command to stop the encoder if it is running. Wait for the command to complete.
- Issue the FWapi command to stop the decoder if it is running. Wait for the command to complete.
- Issue the I2C command to the digitizer to stop emitting VSYNC events.
- Issue the FWapi command to halt the encoder's firmware.
- Sleep for 10ms.
- Issue the FWapi command to halt the decoder's firmware.
- Sleep for 10ms.
- Write 0x00000000 to register 0x2800 to stop the Video Display Module.
- Write 0x00000005 to register 0x2D00 to stop the AO (audio output?).
- Write 0x00000000 to register 0xA064 to ping? the APU.
- Write 0xFFFFFFFE to register 0x9058 to stop the VPU.

- Write 0xFFFFFFFF to register 0x9054 to reset the HW blocks.
- Write 0x00000001 to register 0x9050 to stop the SPU.
- Sleep for 10ms.
- Write 0x0000001A to register 0x07FC to init the Encoder SDRAM's pre-charge.
- Write 0x80000640 to register 0x07F8 to init the Encoder SDRAM's refresh to 1us.
- Write 0x0000001A to register 0x08FC to init the Decoder SDRAM's pre-charge.
- Write 0x80000640 to register 0x08F8 to init the Decoder SDRAM's refresh to 1us.
- Sleep for 512ms. (600ms is recommended)
- Transfer the encoder's firmware image to offset 0 in Encoder memory space.
- Transfer the decoder's firmware image to offset 0 in Decoder memory space.
- Use a read-modify-write operation to Clear bit 0 of register 0x9050 to re-enable the SPU.
- Sleep for 1 second.
- Use a read-modify-write operation to Clear bits 3 and 0 of register 0x9058 to re-enable the VPU.
- Sleep for 1 second.
- Issue status API commands to both firmware images to verify.

4.11.4 How to call the firmware API

The preferred calling convention is known as the firmware mailbox. The mailboxes are basically a fixed length array that serves as the call-stack.

Firmware mailboxes can be located by searching the encoder and decoder memory for a 16 byte signature. That signature will be located on a 256-byte boundary.

Signature:

```
0x78, 0x56, 0x34, 0x12, 0x12, 0x78, 0x56, 0x34,
0x34, 0x12, 0x78, 0x56, 0x56, 0x34, 0x12, 0x78
```

The firmware implements 20 mailboxes of 20 32-bit words. The first 10 are reserved for API calls. The second 10 are used by the firmware for event notification.

Index	Name
0	Flags
1	Command
2	Return value
3	Timeout
4-19	Parameter/Result

The flags are defined in the following table. The direction is from the perspective of the firmware.

Bit	Direction	Purpose
2	O	Firmware has processed the command.
1	I	Driver has finished setting the parameters.
0	I	Driver is using this mailbox.

The command is a 32-bit enumerator. The API specifics may be found in this chapter.

The return value is a 32-bit enumerator. Only two values are currently defined:

- 0=success
- -1=command undefined.

There are 16 parameters/results 32-bit fields. The driver populates these fields with values for all the parameters required by the call. The driver overwrites these fields with result values returned by the call.

The timeout value protects the card from a hung driver thread. If the driver doesn't handle the completed call within the timeout specified, the firmware will reset that mailbox.

To make an API call, the driver iterates over each mailbox looking for the first one available (bit 0 has been cleared). The driver sets that bit, fills in the command enumerator, the timeout value and any required parameters. The driver then sets the parameter ready bit (bit 1). The firmware scans the mailboxes for pending commands, processes them, sets the result code, populates the result value array with that call's return values and sets the call complete bit (bit 2). Once bit 2 is set, the driver should retrieve the results and clear all the flags. If the driver does not perform this task within the time set in the timeout register, the firmware will reset that mailbox.

Event notifications are sent from the firmware to the host. The host tells the firmware which events it is interested in via an API call. That call tells the firmware which notification mailbox to use. The firmware signals the host via an interrupt. Only the 16 Results fields are used, the Flags, Command, Return value and Timeout words are not used.

4.11.5 OSD firmware API description

Note:

this API is part of the decoder firmware, so it's cx23415 only.

CX2341X_OSD_GET_FRAMEBUFFER

Enum: 65/0x41

Description

Return base and length of contiguous OSD memory.

Result[0]

OSD base address

Result[1]

OSD length

CX2341X_OSD_GET_PIXEL_FORMAT

Enum: 66/0x42

Description

Query OSD format

Result[0]

0=8bit index 1=16bit RGB 5:6:5 2=16bit ARGB 1:5:5:5 3=16bit ARGB 1:4:4:4 4=32bit ARGB 8:8:8:8

CX2341X_OSD_SET_PIXEL_FORMAT

Enum: 67/0x43

Description

Assign pixel format

Param[0]

- 0=8bit index
- 1=16bit RGB 5:6:5
- 2=16bit ARGB 1:5:5:5
- 3=16bit ARGB 1:4:4:4
- 4=32bit ARGB 8:8:8:8

CX2341X_OSD_GET_STATE

Enum: 68/0x44

Description

Query OSD state

Result[0]

- Bit 0 0=off, 1=on
- Bits 1:2 alpha control
- Bits 3:5 pixel format

CX2341X_OSD_SET_STATE

Enum: 69/0x45

Description

OSD switch

Param[0]

0=off, 1=on

CX2341X_OSD_GET_OSD_COORDS

Enum: 70/0x46

Description

Retrieve coordinates of OSD area blended with video

Result[0]

OSD buffer address

Result[1]

Stride in pixels

Result[2]

Lines in OSD buffer

Result[3]

Horizontal offset in buffer

Result[4]

Vertical offset in buffer

CX2341X_OSD_SET_OSD_COORDS

Enum: 71/0x47

Description

Assign the coordinates of the OSD area to blend with video

Param[0]

buffer address

Param[1]

buffer stride in pixels

Param[2]

lines in buffer

Param[3]

horizontal offset

Param[4]

vertical offset

CX2341X_OSD_GET_SCREEN_COORDS

Enum: 72/0x48

Description

Retrieve OSD screen area coordinates

Result[0]

top left horizontal offset

Result[1]

top left vertical offset

Result[2]

bottom right horizontal offset

Result[3]

bottom right vertical offset

CX2341X_OSD_SET_SCREEN_COORDS

Enum: 73/0x49

Description

Assign the coordinates of the screen area to blend with video

Param[0]

top left horizontal offset

Param[1]

top left vertical offset

Param[2]

bottom left horizontal offset

Param[3]

bottom left vertical offset

CX2341X_OSD_GET_GLOBAL_ALPHA

Enum: 74/0x4A

Description

Retrieve OSD global alpha

Result[0]

global alpha: 0=off, 1=on

Result[1]

bits 0:7 global alpha

CX2341X_OSD_SET_GLOBAL_ALPHA

Enum: 75/0x4B

Description

Update global alpha

Param[0]

global alpha: 0=off, 1=on

Param[1]

global alpha (8 bits)

Param[2]

local alpha: 0=on, 1=off

CX2341X_OSD_SET_BLEND_COORDS

Enum: 78/0x4C

Description

Move start of blending area within display buffer

Param[0]

horizontal offset in buffer

Param[1]

vertical offset in buffer

CX2341X_OSD_GET_FLICKER_STATE

Enum: 79/0x4F

Description

Retrieve flicker reduction module state

Result[0]

flicker state: 0=off, 1=on

CX2341X_OSD_SET_FLICKER_STATE

Enum: 80/0x50

Description

Set flicker reduction module state

Param[0]

State: 0=off, 1=on

CX2341X_OSD_BLT_COPY

Enum: 82/0x52

Description

BLT copy

Param[0]

```
'0000'  zero
'0001' ~destination AND ~source
'0010' ~destination AND  source
'0011' ~destination
'0100'  destination AND ~source
'0101'                  ~source
'0110'  destination XOR  source
'0111' ~destination OR  ~source
'1000' ~destination AND ~source
'1001'  destination XNOR source
'1010'                   source
'1011' ~destination OR   source
'1100'  destination
'1101'  destination OR  ~source
'1110'  destination OR   source
'1111'  one
```

Param[1]

Resulting alpha blending

- '01' source_alpha
- '10' destination_alpha
- '11' source_alpha*destination_alpha+1 (zero if both source and destination alpha are zero)

Param[2]

```
'00' output_pixel = source_pixel

'01' if source_alpha=0:
         output_pixel = destination_pixel
     if 256 > source_alpha > 1:
         output_pixel = ((source_alpha + 1)*source_pixel +
                         (255 - source_alpha)*destination_pixel)/256

'10' if destination_alpha=0:
         output_pixel = source_pixel
      if 255 > destination_alpha > 0:
         output_pixel = ((255 - destination_alpha)*source_pixel +
                         (destination_alpha + 1)*destination_pixel)/256

'11' if source_alpha=0:
         source_temp = 0
     if source_alpha=255:
         source_temp = source_pixel*256
     if 255 > source_alpha > 0:
         source_temp = source_pixel*(source_alpha + 1)
     if destination_alpha=0:
         destination_temp = 0
     if destination_alpha=255:
         destination_temp = destination_pixel*256
     if 255 > destination_alpha > 0:
         destination_temp = destination_pixel*(destination_alpha + 1)
     output_pixel = (source_temp + destination_temp)/256
```

Param[3]

width

Param[4]

height

Param[5]

destination pixel mask

Param[6]

destination rectangle start address

Param[7]

destination stride in dwords

Param[8]

source stride in dwords

Param[9]

source rectangle start address

CX2341X_OSD_BLT_FILL

Enum: 83/0x53

Description

BLT fill color

Param[0]

Same as Param[0] on API 0x52

Param[1]

Same as Param[1] on API 0x52

Param[2]

Same as Param[2] on API 0x52

Param[3]

width

Param[4]

height

Param[5]

destination pixel mask

Param[6]

destination rectangle start address

Param[7]

destination stride in dwords

Param[8]

color fill value

CX2341X_OSD_BLT_TEXT

Enum: 84/0x54

Description

BLT for 8 bit alpha text source

Param[0]

Same as Param[0] on API 0x52

Param[1]

Same as Param[1] on API 0x52

Param[2]

Same as Param[2] on API 0x52

Param[3]

width

Param[4]

height

Param[5]

destination pixel mask

Param[6]

destination rectangle start address

Param[7]

destination stride in dwords

Param[8]

source stride in dwords

Param[9]

source rectangle start address

Param[10]

color fill value

CX2341X_OSD_SET_FRAMEBUFFER_WINDOW

Enum: 86/0x56

Description

Positions the main output window on the screen. The coordinates must be such that the entire window fits on the screen.

Param[0]

window width

Param[1]

window height

Param[2]

top left window corner horizontal offset

Param[3]

top left window corner vertical offset

CX2341X_OSD_SET_CHROMA_KEY

Enum: 96/0x60

Description

Chroma key switch and color

Param[0]

state: 0=off, 1=on

Param[1]

color

CX2341X_OSD_GET_ALPHA_CONTENT_INDEX

Enum: 97/0x61

Description

Retrieve alpha content index

Result[0]

alpha content index, Range 0:15

CX2341X_OSD_SET_ALPHA_CONTENT_INDEX

Enum: 98/0x62

Description

Assign alpha content index

Param[0]

alpha content index, range 0:15

4.11.6 Encoder firmware API description

CX2341X_ENC_PING_FW

Enum: 128/0x80

Description

Does nothing. Can be used to check if the firmware is responding.

CX2341X_ENC_START_CAPTURE

Enum: 129/0x81

Description

Commences the capture of video, audio and/or VBI data. All encoding parameters must be initialized prior to this API call. Captures frames continuously or until a predefined number of frames have been captured.

Param[0]

Capture stream type:

- 0=MPEG
- 1=Raw
- 2=Raw passthrough
- 3=VBI

Param[1]

Bitmask:

- Bit 0 when set, captures YUV
- Bit 1 when set, captures PCM audio
- Bit 2 when set, captures VBI (same as param[0]=3)
- Bit 3 when set, the capture destination is the decoder (same as param[0]=2)
- Bit 4 when set, the capture destination is the host

Note:

this parameter is only meaningful for RAW capture type.

CX2341X_ENC_STOP_CAPTURE

Enum: 130/0x82

Description

Ends a capture in progress

Param[0]

- 0=stop at end of GOP (generates IRQ)
- 1=stop immediate (no IRQ)

Param[1]

Stream type to stop, see param[0] of API 0x81

Param[2]

Subtype, see param[1] of API 0x81

CX2341X_ENC_SET_AUDIO_ID

Enum: 137/0x89

Description

Assigns the transport stream ID of the encoded audio stream

Param[0]

Audio Stream ID

CX2341X_ENC_SET_VIDEO_ID

Enum: 139/0x8B

Description

Set video transport stream ID

Param[0]

Video stream ID

CX2341X_ENC_SET_PCR_ID

Enum: 141/0x8D

Description

Assigns the transport stream ID for PCR packets

Param[0]

PCR Stream ID

CX2341X_ENC_SET_FRAME_RATE

Enum: 143/0x8F

Description

Set video frames per second. Change occurs at start of new GOP.

Param[0]

- 0=30fps
- 1=25fps

CX2341X_ENC_SET_FRAME_SIZE

Enum: 145/0x91

Description

Select video stream encoding resolution.

Param[0]

Height in lines. Default 480

Param[1]

Width in pixels. Default 720

CX2341X_ENC_SET_BIT_RATE

Enum: 149/0x95

Description

Assign average video stream bitrate.

Param[0]

0=variable bitrate, 1=constant bitrate

Param[1]

bitrate in bits per second

Param[2]

peak bitrate in bits per second, divided by 400

Param[3]

Mux bitrate in bits per second, divided by 400. May be 0 (default).

Param[4]

Rate Control VBR Padding

Param[5]

VBV Buffer used by encoder

Note:

1. *Param[3] and Param[4] seem to be always 0*
2. *Param[5] doesn't seem to be used.*

CX2341X_ENC_SET_GOP_PROPERTIES

Enum: 151/0x97

Description

Setup the GOP structure

Param[0]

GOP size (maximum is 34)

Param[1]

Number of B frames between the I and P frame, plus 1. For example: IBBPBBPBBPBB –> GOP size: 12, number of B frames: 2+1 = 3

Note:

GOP size must be a multiple of (B-frames + 1).

CX2341X_ENC_SET_ASPECT_RATIO

Enum: 153/0x99

Description

Sets the encoding aspect ratio. Changes in the aspect ratio take effect at the start of the next GOP.

Param[0]

- '0000' forbidden
- '0001' 1:1 square
- '0010' 4:3
- '0011' 16:9
- '0100' 2.21:1
- '0101' to '1111' reserved

CX2341X_ENC_SET_DNR_FILTER_MODE

Enum: 155/0x9B

Description

Assign Dynamic Noise Reduction operating mode

Param[0]

Bit0: Spatial filter, set=auto, clear=manual Bit1: Temporal filter, set=auto, clear=manual

Param[1]

Median filter:

- 0=Disabled
- 1=Horizontal
- 2=Vertical
- 3=Horiz/Vert
- 4=Diagonal

CX2341X_ENC_SET_DNR_FILTER_PROPS

Enum: 157/0x9D

Description

These Dynamic Noise Reduction filter values are only meaningful when the respective filter is set to "manual" (See API 0x9B)

Param[0]

Spatial filter: default 0, range 0:15

Param[1]

Temporal filter: default 0, range 0:31

CX2341X_ENC_SET_CORING_LEVELS

Enum: 159/0x9F

Description

Assign Dynamic Noise Reduction median filter properties.

Param[0]

Threshold above which the luminance median filter is enabled. Default: 0, range 0:255

Param[1]

Threshold below which the luminance median filter is enabled. Default: 255, range 0:255

Param[2]

Threshold above which the chrominance median filter is enabled. Default: 0, range 0:255

Param[3]

Threshold below which the chrominance median filter is enabled. Default: 255, range 0:255

CX2341X_ENC_SET_SPATIAL_FILTER_TYPE

Enum: 161/0xA1

Description

Assign spatial prefilter parameters

Param[0]

Luminance filter

- 0=Off
- 1=1D Horizontal
- 2=1D Vertical
- 3=2D H/V Separable (default)
- 4=2D Symmetric non-separable

Param[1]

Chrominance filter

- 0=Off
- 1=1D Horizontal (default)

CX2341X_ENC_SET_VBI_LINE

Enum: 183/0xB7

Description

Selects VBI line number.

Param[0]

- Bits 0:4 line number
- Bit 31 0=top_field, 1=bottom_field
- Bits 0:31 all set specifies “all lines”

Param[1]

VBI line information features: 0=disabled, 1=enabled

Param[2]

Slicing: 0=None, 1=Closed Caption Almost certainly not implemented. Set to 0.

Param[3]

Luminance samples in this line. Almost certainly not implemented. Set to 0.

Param[4]

Chrominance samples in this line Almost certainly not implemented. Set to 0.

CX2341X_ENC_SET_STREAM_TYPE

Enum: 185/0xB9

Description

Assign stream type

Note:

Transport stream is not working in recent firmwares. And in older firmwares the timestamps in the TS seem to be unreliable.

Param[0]

- 0=Program stream
- 1=Transport stream
- 2=MPEG1 stream
- 3=PES A/V stream
- 5=PES Video stream
- 7=PES Audio stream
- 10=DVD stream
- 11=VCD stream
- 12=SVCD stream
- 13=DVD_S1 stream
- 14=DVD_S2 stream

CX2341X_ENC_SET_OUTPUT_PORT

Enum: 187/0xBB

Description

Assign stream output port. Normally 0 when the data is copied through the PCI bus (DMA), and 1 when the data is streamed to another chip (pvrusb and cx88-blackbird).

Param[0]

- 0=Memory (default)
- 1=Streaming
- 2=Serial

Param[1]

Unknown, but leaving this to 0 seems to work best. Indications are that this might have to do with USB support, although passing anything but 0 only breaks things.

CX2341X_ENC_SET_AUDIO_PROPERTIES

Enum: 189/0xBD

Description

Set audio stream properties, may be called while encoding is in progress.

Note:

All bitfields are consistent with ISO11172 documentation except bits 2:3 which ISO docs define as:

- *'11' Layer I*
- *'10' Layer II*
- *'01' Layer III*
- *'00' Undefined*

This discrepancy may indicate a possible error in the documentation. Testing indicated that only Layer II is actually working, and that the minimum bitrate should be 192 kbps.

Param[0]

Bitmask:

```
0:1  '00' 44.1Khz
     '01' 48Khz
     '10' 32Khz
     '11' reserved

2:3  '01'=Layer I
     '10'=Layer II

4:7  Bitrate:
          Index | Layer I     | Layer II
          ------+-------------+------------
         '0000' | free format | free format
         '0001' |  32 kbit/s  |  32 kbit/s
         '0010' |  64 kbit/s  |  48 kbit/s
         '0011' |  96 kbit/s  |  56 kbit/s
         '0100' | 128 kbit/s  |  64 kbit/s
         '0101' | 160 kbit/s  |  80 kbit/s
         '0110' | 192 kbit/s  |  96 kbit/s
         '0111' | 224 kbit/s  | 112 kbit/s
         '1000' | 256 kbit/s  | 128 kbit/s
         '1001' | 288 kbit/s  | 160 kbit/s
         '1010' | 320 kbit/s  | 192 kbit/s
         '1011' | 352 kbit/s  | 224 kbit/s
         '1100' | 384 kbit/s  | 256 kbit/s
         '1101' | 416 kbit/s  | 320 kbit/s
         '1110' | 448 kbit/s  | 384 kbit/s

     .. note::

             For Layer II, not all combinations of total bitrate
             and mode are allowed. See ISO11172-3 3-Annex B,
             Table 3-B.2

8:9  '00'=Stereo
     '01'=JointStereo
     '10'=Dual
     '11'=Mono

     .. note::

             The cx23415 cannot decode Joint Stereo properly.
```

```
10:11 Mode Extension used in joint_stereo mode.
      In Layer I and II they indicate which subbands are in
      intensity_stereo. All other subbands are coded in stereo.
          '00' subbands 4-31 in intensity_stereo, bound==4
          '01' subbands 8-31 in intensity_stereo, bound==8
          '10' subbands 12-31 in intensity_stereo, bound==12
          '11' subbands 16-31 in intensity_stereo, bound==16

12:13 Emphasis:
          '00' None
          '01' 50/15uS
          '10' reserved
          '11' CCITT J.17

14    CRC:
          '0' off
          '1' on

15    Copyright:
          '0' off
          '1' on

16    Generation:
          '0' copy
          '1' original
```

CX2341X_ENC_HALT_FW

Enum: 195/0xC3

Description

The firmware is halted and no further API calls are serviced until the firmware is uploaded again.

CX2341X_ENC_GET_VERSION

Enum: 196/0xC4

Description

Returns the version of the encoder firmware.

Result[0]

Version bitmask: - Bits 0:15 build - Bits 16:23 minor - Bits 24:31 major

CX2341X_ENC_SET_GOP_CLOSURE

Enum: 197/0xC5

Description

Assigns the GOP open/close property.

Param[0]

- 0=Open
- 1=Closed

CX2341X_ENC_GET_SEQ_END

Enum: 198/0xC6

Description

Obtains the sequence end code of the encoder's buffer. When a capture is started a number of interrupts are still generated, the last of which will have Result[0] set to 1 and Result[1] will contain the size of the buffer.

Result[0]

State of the transfer (1 if last buffer)

Result[1]

If Result[0] is 1, this contains the size of the last buffer, undefined otherwise.

CX2341X_ENC_SET_PGM_INDEX_INFO

Enum: 199/0xC7

Description

Sets the Program Index Information. The information is stored as follows:

```
struct info {
        u32 length;             // Length of this frame
        u32 offset_low;         // Offset in the file of the
        u32 offset_high;        // start of this frame
        u32 mask1;              // Bits 0-2 are the type mask:
                                // 1=I, 2=P, 4=B
                                // 0=End of Program Index, other fields
                                //   are invalid.
        u32 pts;                // The PTS of the frame
        u32 mask2;              // Bit 0 is bit 32 of the pts.
};
u32 table_ptr;
struct info index[400];
```

The table_ptr is the encoder memory address in the table were *new* entries will be written.

Note:

This is a ringbuffer, so the table_ptr will wraparound.

Param[0]

Picture Mask: - 0=No index capture - 1=I frames - 3=I,P frames - 7=I,P,B frames
(Seems to be ignored, it always indexes I, P and B frames)

Param[1]

Elements requested (up to 400)

Result[0]

Offset in the encoder memory of the start of the table.

Result[1]

Number of allocated elements up to a maximum of Param[1]

CX2341X_ENC_SET_VBI_CONFIG

Enum: 200/0xC8

Description

Configure VBI settings

Param[0]

Bitmap:

```
0    Mode '0' Sliced, '1' Raw
1:3  Insertion:
         '000' insert in extension & user data
         '001' insert in private packets
         '010' separate stream and user data
         '111' separate stream and private data
8:15 Stream ID (normally 0xBD)
```

Param[1]

Frames per interrupt (max 8). Only valid in raw mode.

Param[2]

Total raw VBI frames. Only valid in raw mode.

Param[3]

Start codes

Param[4]

Stop codes

Param[5]

Lines per frame

Param[6]

Byte per line

Result[0]

Observed frames per interrupt in raw mode only. Rage 1 to Param[1]

Result[1]

Observed number of frames in raw mode. Range 1 to Param[2]

Result[2]

Memory offset to start or raw VBI data

CX2341X_ENC_SET_DMA_BLOCK_SIZE

Enum: 201/0xC9

Description

Set DMA transfer block size

Param[0]

DMA transfer block size in bytes or frames. When unit is bytes, supported block sizes are 2^7, 2^8 and 2^9 bytes.

Param[1]

Unit: 0=bytes, 1=frames

CX2341X_ENC_GET_PREV_DMA_INFO_MB_10

Enum: 202/0xCA

Description

Returns information on the previous DMA transfer in conjunction with bit 27 of the interrupt mask. Uses mailbox 10.

Result[0]

Type of stream

Result[1]

Address Offset

Result[2]

Maximum size of transfer

CX2341X_ENC_GET_PREV_DMA_INFO_MB_9

Enum: 203/0xCB

Description

Returns information on the previous DMA transfer in conjunction with bit 27 or 18 of the interrupt mask. Uses mailbox 9.

Result[0]

Status bits: - 0 read completed - 1 write completed - 2 DMA read error - 3 DMA write error - 4 Scatter-Gather array error

Result[1]

DMA type

Result[2]

Presentation Time Stamp bits 0..31

Result[3]

Presentation Time Stamp bit 32

CX2341X_ENC_SCHED_DMA_TO_HOST

Enum: 204/0xCC

Description

Setup DMA to host operation

Param[0]

Memory address of link list

Param[1]

Length of link list (wtf: what units ???)

Param[2]

DMA type (0=MPEG)

CX2341X_ENC_INITIALIZE_INPUT

Enum: 205/0xCD

Description

Initializes the video input

CX2341X_ENC_SET_FRAME_DROP_RATE

Enum: 208/0xD0

Description

For each frame captured, skip specified number of frames.

Param[0]

Number of frames to skip

CX2341X_ENC_PAUSE_ENCODER

Enum: 210/0xD2

Description

During a pause condition, all frames are dropped instead of being encoded.

Param[0]

- 0=Pause encoding
- 1=Continue encoding

CX2341X_ENC_REFRESH_INPUT

Enum: 211/0xD3

Description

Refreshes the video input

CX2341X_ENC_SET_COPYRIGHT

Enum: 212/0xD4

Description

Sets stream copyright property

Param[0]

- 0=Stream is not copyrighted
- 1=Stream is copyrighted

CX2341X_ENC_SET_EVENT_NOTIFICATION

Enum: 213/0xD5

Description

Setup firmware to notify the host about a particular event. Host must unmask the interrupt bit.

Param[0]

Event (0=refresh encoder input)

Param[1]

Notification 0=disabled 1=enabled

Param[2]

Interrupt bit

Param[3]

Mailbox slot, -1 if no mailbox required.

CX2341X_ENC_SET_NUM_VSYNC_LINES

Enum: 214/0xD6

Description

Depending on the analog video decoder used, this assigns the number of lines for field 1 and 2.

Param[0]

Field 1 number of lines: - 0x00EF for SAA7114 - 0x00F0 for SAA7115 - 0x0105 for Micronas

Param[1]

Field 2 number of lines: - 0x00EF for SAA7114 - 0x00F0 for SAA7115 - 0x0106 for Micronas

CX2341X_ENC_SET_PLACEHOLDER

Enum: 215/0xD7

Description

Provides a mechanism of inserting custom user data in the MPEG stream.

Param[0]

- 0=extension & user data
- 1=private packet with stream ID 0xBD

Param[1]

Rate at which to insert data, in units of frames (for private packet) or GOPs (for ext. & user data)

Param[2]

Number of data DWORDs (below) to insert

Param[3]

Custom data 0

Param[4]

Custom data 1

Param[5]

Custom data 2

Param[6]

Custom data 3

Param[7]

Custom data 4

Param[8]

Custom data 5

Param[9]

Custom data 6

Param[10]

Custom data 7

Param[11]

Custom data 8

CX2341X_ENC_MUTE_VIDEO

Enum: 217/0xD9

Description

Video muting

Param[0]

Bit usage:

```
 0      '0'=video not muted
        '1'=video muted, creates frames with the YUV color defined below
 1:7    Unused
 8:15   V chrominance information
16:23   U chrominance information
24:31   Y luminance information
```

CX2341X_ENC_MUTE_AUDIO

Enum: 218/0xDA

Description

Audio muting

Param[0]

- 0=audio not muted
- 1=audio muted (produces silent mpeg audio stream)

CX2341X_ENC_SET_VERT_CROP_LINE

Enum: 219/0xDB

Description

Something to do with 'Vertical Crop Line'

Param[0]

If saa7114 and raw VBI capture and 60 Hz, then set to 10001. Else 0.

CX2341X_ENC_MISC

Enum: 220/0xDC

Description

Miscellaneous actions. Not known for 100% what it does. It's really a sort of ioctl call. The first parameter is a command number, the second the value.

Param[0]

Command number:

```
 1=set initial SCR value when starting encoding (works).
 2=set quality mode (apparently some test setting).
 3=setup advanced VIM protection handling.
   Always 1 for the cx23416 and 0 for cx23415.
 4=generate DVD compatible PTS timestamps
 5=USB flush mode
 6=something to do with the quantization matrix
 7=set navigation pack insertion for DVD: adds 0xbf (private stream 2)
   packets to the MPEG. The size of these packets is 2048 bytes (including
   the header of 6 bytes: 0x000001bf + length). The payload is zeroed and
   it is up to the application to fill them in. These packets are apparently
   inserted every four frames.
 8=enable scene change detection (seems to be a failure)
 9=set history parameters of the video input module
10=set input field order of VIM
11=set quantization matrix
12=reset audio interface after channel change or input switch (has no argument).
   Needed for the cx2584x, not needed for the mspx4xx, but it doesn't seem to
```

```
   do any harm calling it regardless.
13=set audio volume delay
14=set audio delay
```

Param[1]

Command value.

4.11.7 Decoder firmware API description

Note:

this API is part of the decoder firmware, so it's cx23415 only.

CX2341X_DEC_PING_FW

Enum: 0/0x00

Description

This API call does nothing. It may be used to check if the firmware is responding.

CX2341X_DEC_START_PLAYBACK

Enum: 1/0x01

Description

Begin or resume playback.

Param[0]

0 based frame number in GOP to begin playback from.

Param[1]

Specifies the number of muted audio frames to play before normal audio resumes. (This is not implemented in the firmware, leave at 0)

CX2341X_DEC_STOP_PLAYBACK

Enum: 2/0x02

Description

Ends playback and clears all decoder buffers. If PTS is not zero, playback stops at specified PTS.

Param[0]

Display 0=last frame, 1=black

Note:

this takes effect immediately, so if you want to wait for a PTS, then use '0', otherwise the screen goes to black at once. You can call this later (even if there is no playback) with a 1 value to set the screen to black.

Param[1]

PTS low

Param[2]

PTS high

CX2341X_DEC_SET_PLAYBACK_SPEED

Enum: 3/0x03

Description

Playback stream at speed other than normal. There are two modes of operation:

- Smooth: host transfers entire stream and firmware drops unused frames.
- Coarse: host drops frames based on indexing as required to achieve desired speed.

Param[0]

```
Bitmap:
    0:7  0 normal
         1 fast only "1.5 times"
         n nX fast, 1/nX slow
    30   Framedrop:
             '0' during 1.5 times play, every other B frame is dropped
             '1' during 1.5 times play, stream is unchanged (bitrate
                 must not exceed 8mbps)
    31   Speed:
             '0' slow
             '1' fast
```

Note:

n is limited to 2. Anything higher does not result in faster playback. Instead the host should start dropping frames.

Param[1]

Direction: 0=forward, 1=reverse

Note:

to make reverse playback work you have to write full GOPs in reverse order.

Param[2]

```
Picture mask:
    1=I frames
    3=I, P frames
    7=I, P, B frames
```

Param[3]

B frames per GOP (for reverse play only)

Note:

for reverse playback the Picture Mask should be set to I or I, P. Adding B frames to the mask will result in corrupt video. This field has to be set to the correct value in order to keep the timing correct.

Param[4]

Mute audio: 0=disable, 1=enable

Param[5]

Display 0=frame, 1=field

Param[6]

Specifies the number of muted audio frames to play before normal audio resumes. (Not implemented in the firmware, leave at 0)

CX2341X_DEC_STEP_VIDEO

Enum: 5/0x05

Description

Each call to this API steps the playback to the next unit defined below in the current playback direction.

Param[0]

0=frame, 1=top field, 2=bottom field

CX2341X_DEC_SET_DMA_BLOCK_SIZE

Enum: 8/0x08

Description

Set DMA transfer block size. Counterpart to API 0xC9

Param[0]

DMA transfer block size in bytes. A different size may be specified when issuing the DMA transfer command.

CX2341X_DEC_GET_XFER_INFO

Enum: 9/0x09

Description

This API call may be used to detect an end of stream condition.

Result[0]

Stream type

Result[1]

Address offset

Result[2]

Maximum bytes to transfer

Result[3]

Buffer fullness

CX2341X_DEC_GET_DMA_STATUS

Enum: 10/0x0A

Description

Status of the last DMA transfer

Result[0]

Bit 1 set means transfer complete Bit 2 set means DMA error Bit 3 set means linked list error

Result[1]

DMA type: 0=MPEG, 1=OSD, 2=YUV

CX2341X_DEC_SCHED_DMA_FROM_HOST

Enum: 11/0x0B

Description

Setup DMA from host operation. Counterpart to API 0xCC

Param[0]

Memory address of link list

Param[1]

Total # of bytes to transfer

Param[2]

DMA type (0=MPEG, 1=OSD, 2=YUV)

CX2341X_DEC_PAUSE_PLAYBACK

Enum: 13/0x0D

Description

Freeze playback immediately. In this mode, when internal buffers are full, no more data will be accepted and data request IRQs will be masked.

Param[0]

Display: 0=last frame, 1=black

CX2341X_DEC_HALT_FW

Enum: 14/0x0E

Description

The firmware is halted and no further API calls are serviced until the firmware is uploaded again.

CX2341X_DEC_SET_STANDARD

Enum: 16/0x10

Description

Selects display standard

Param[0]

0=NTSC, 1=PAL

CX2341X_DEC_GET_VERSION

Enum: 17/0x11

Description

Returns decoder firmware version information

Result[0]

Version bitmask:

- Bits 0:15 build
- Bits 16:23 minor
- Bits 24:31 major

CX2341X_DEC_SET_STREAM_INPUT

Enum: 20/0x14

Description

Select decoder stream input port

Param[0]

0=memory (default), 1=streaming

CX2341X_DEC_GET_TIMING_INFO

Enum: 21/0x15

Description

Returns timing information from start of playback

Result[0]

Frame count by decode order

Result[1]

Video PTS bits 0:31 by display order

Result[2]

Video PTS bit 32 by display order

Result[3]

SCR bits 0:31 by display order

Result[4]

SCR bit 32 by display order

CX2341X_DEC_SET_AUDIO_MODE

Enum: 22/0x16

Description

Select audio mode

Param[0]

Dual mono mode action 0=Stereo, 1=Left, 2=Right, 3=Mono, 4=Swap, -1=Unchanged

Param[1]

Stereo mode action: 0=Stereo, 1=Left, 2=Right, 3=Mono, 4=Swap, -1=Unchanged

CX2341X_DEC_SET_EVENT_NOTIFICATION

Enum: 23/0x17

Description

Setup firmware to notify the host about a particular event. Counterpart to API 0xD5

Param[0]

Event:

- 0=Audio mode change between mono, (joint) stereo and dual channel.
- 3=Decoder started
- 4=Unknown: goes off 10-15 times per second while decoding.
- 5=Some sync event: goes off once per frame.

Param[1]

Notification 0=disabled, 1=enabled

Param[2]

Interrupt bit

Param[3]

Mailbox slot, -1 if no mailbox required.

CX2341X_DEC_SET_DISPLAY_BUFFERS

Enum: 24/0x18

Description

Number of display buffers. To decode all frames in reverse playback you must use nine buffers.

Param[0]

0=six buffers, 1=nine buffers

CX2341X_DEC_EXTRACT_VBI

Enum: 25/0x19

Description

Extracts VBI data

Param[0]

0=extract from extension & user data, 1=extract from private packets

Result[0]

VBI table location

Result[1]

VBI table size

CX2341X_DEC_SET_DECODER_SOURCE

Enum: 26/0x1A

Description

Selects decoder source. Ensure that the parameters passed to this API match the encoder settings.

Param[0]

Mode: 0=MPEG from host, 1=YUV from encoder, 2=YUV from host

Param[1]

YUV picture width

Param[2]

YUV picture height

Param[3]

Bitmap: see Param[0] of API 0xBD

CX2341X_DEC_SET_PREBUFFERING

Enum: 30/0x1E

Description

Decoder prebuffering, when enabled up to 128KB are buffered for streams <8mpbs or 640KB for streams >8mbps

Param[0]

0=off, 1=on

4.11.8 PVR350 Video decoder registers 0x02002800 -> 0x02002B00

Author: Ian Armstrong <ian@iarmst.demon.co.uk>

Version: v0.4

Date: 12 March 2007

This list has been worked out through trial and error. There will be mistakes and omissions. Some registers have no obvious effect so it's hard to say what they do, while others interact with each other, or require a certain load sequence. Horizontal filter setup is one example, with six registers working in unison and requiring a certain load sequence to correctly configure. The indexed colour palette is much easier to set at just two registers, but again it requires a certain load sequence.

Some registers are fussy about what they are set to. Load in a bad value & the decoder will fail. A firmware reload will often recover, but sometimes a reset is required. For registers containing size information, setting them to 0 is generally a bad idea. For other control registers i.e. 2878, you'll only find out what values are bad when it hangs.

```
------------------------------------------------------------------------------
2800
bit 0
        Decoder enable
        0 = disable
        1 = enable
------------------------------------------------------------------------------
2804
bits 0:31
        Decoder horizontal Y alias register 1
---------------
2808
bits 0:31
        Decoder horizontal Y alias register 2
---------------
280C
bits 0:31
        Decoder horizontal Y alias register 3
---------------
2810
bits 0:31
        Decoder horizontal Y alias register 4
---------------
2814
bits 0:31
        Decoder horizontal Y alias register 5
---------------
2818
bits 0:31
        Decoder horizontal Y alias trigger

These six registers control the horizontal aliasing filter for the Y plane.
The first five registers must all be loaded before accessing the trigger
(2818), as this register actually clocks the data through for the first
five.

To correctly program set the filter, this whole procedure must be done 16
times. The actual register contents are copied from a lookup-table in the
firmware which contains 4 different filter settings.

------------------------------------------------------------------------------
281C
bits 0:31
        Decoder horizontal UV alias register 1
---------------
2820
bits 0:31
        Decoder horizontal UV alias register 2
---------------
2824
bits 0:31
        Decoder horizontal UV alias register 3
```

```
---------------
2828
bits 0:31
        Decoder horizontal UV alias register 4
---------------
282C
bits 0:31
        Decoder horizontal UV alias register 5
---------------
2830
bits 0:31
        Decoder horizontal UV alias trigger

These six registers control the horizontal aliasing for the UV plane.
Operation is the same as the Y filter, with 2830 being the trigger
register.

--------------------------------------------------------------------------------
2834
bits 0:15
        Decoder Y source width in pixels

bits 16:31
        Decoder Y destination width in pixels
---------------
2838
bits 0:15
        Decoder UV source width in pixels

bits 16:31
        Decoder UV destination width in pixels

NOTE: For both registers, the resulting image must be fully visible on
screen. If the image exceeds the right edge both the source and destination
size must be adjusted to reflect the visible portion. For the source width,
you must take into account the scaling when calculating the new value.
--------------------------------------------------------------------------------

283C
bits 0:31
        Decoder Y horizontal scaling
                Normally = Reg 2854 >> 2
---------------
2840
bits 0:31
        Decoder ?? unknown - horizontal scaling
        Usually 0x00080514
---------------
2844
bits 0:31
        Decoder UV horizontal scaling
        Normally = Reg 2854 >> 2
---------------
2848
bits 0:31
        Decoder ?? unknown - horizontal scaling
        Usually 0x00100514
---------------
284C
bits 0:31
        Decoder ?? unknown - Y plane
        Usually 0x00200020
---------------
```

```
2850
bits 0:31
        Decoder ?? unknown - UV plane
        Usually 0x00200020
---------------
2854
bits 0:31
        Decoder 'master' value for horizontal scaling
---------------
2858
bits 0:31
        Decoder ?? unknown
        Usually 0
---------------
285C
bits 0:31
        Decoder ?? unknown
        Normally = Reg 2854 >> 1
---------------
2860
bits 0:31
        Decoder ?? unknown
        Usually 0
---------------
2864
bits 0:31
        Decoder ?? unknown
        Normally = Reg 2854 >> 1
---------------
2868
bits 0:31
        Decoder ?? unknown
        Usually 0

Most of these registers either control horizontal scaling, or appear linked
to it in some way. Register 2854 contains the 'master' value & the other
registers can be calculated from that one. You must also remember to
correctly set the divider in Reg 2874.

To enlarge:
        Reg 2854 = (source_width * 0x00200000) / destination_width
        Reg 2874 = No divide

To reduce from full size down to half size:
        Reg 2854 = (source_width/2 * 0x00200000) / destination width
        Reg 2874 = Divide by 2

To reduce from half size down to quarter size:
        Reg 2854 = (source_width/4 * 0x00200000) / destination width
        Reg 2874 = Divide by 4

The result is always rounded up.

-------------------------------------------------------------------------------
286C
bits 0:15
        Decoder horizontal Y buffer offset

bits 15:31
        Decoder horizontal UV buffer offset

Offset into the video image buffer. If the offset is gradually incremented,
the on screen image will move left & wrap around higher up on the right.
```

```
--------------------------------------------------------------------------------
2870
bits 0:15
        Decoder horizontal Y output offset

bits 16:31
        Decoder horizontal UV output offset

Offsets the actual video output. Controls output alignment of the Y & UV
planes. The higher the value, the greater the shift to the left. Use
reg 2890 to move the image right.

--------------------------------------------------------------------------------
2874
bits 0:1
        Decoder horizontal Y output size divider
        00 = No divide
        01 = Divide by 2
        10 = Divide by 3

bits 4:5
        Decoder horizontal UV output size divider
        00 = No divide
        01 = Divide by 2
        10 = Divide by 3

bit 8
        Decoder ?? unknown
        0 = Normal
        1 = Affects video output levels

bit 16
        Decoder ?? unknown
        0 = Normal
        1 = Disable horizontal filter

--------------------------------------------------------------------------------
2878
bit 0
        ?? unknown

bit 1
        osd on/off
        0 = osd off
        1 = osd on

bit 2
        Decoder + osd video timing
        0 = NTSC
        1 = PAL

bits 3:4
        ?? unknown

bit 5
        Decoder + osd
        Swaps upper & lower fields

--------------------------------------------------------------------------------
287C
bits 0:10
        Decoder & osd ?? unknown
```

```
        Moves entire screen horizontally. Starts at 0x005 with the screen
        shifted heavily to the right. Incrementing in steps of 0x004 will
        gradually shift the screen to the left.

bits 11:31
        ?? unknown

Normally contents are 0x00101111 (NTSC) or 0x1010111d (PAL)

--------------------------------------------------------------------------------
2880  --------    ?? unknown
2884  --------    ?? unknown
--------------------------------------------------------------------------------
2888
bit 0
        Decoder + osd ?? unknown
        0 = Normal
        1 = Misaligned fields (Correctable through 289C & 28A4)

bit 4
        ?? unknown

bit 8
        ?? unknown

Warning: Bad values will require a firmware reload to recover.
                Known to be bad are 0x000,0x011,0x100,0x111
--------------------------------------------------------------------------------
288C
bits 0:15
        osd ?? unknown
        Appears to affect the osd position stability. The higher the value the
        more unstable it becomes. Decoder output remains stable.

bits 16:31
        osd ?? unknown
        Same as bits 0:15

--------------------------------------------------------------------------------
2890
bits 0:11
        Decoder output horizontal offset.

Horizontal offset moves the video image right. A small left shift is
possible, but it's better to use reg 2870 for that due to its greater
range.

NOTE: Video corruption will occur if video window is shifted off the right
edge. To avoid this read the notes for 2834 & 2838.
--------------------------------------------------------------------------------
2894
bits 0:23
        Decoder output video surround colour.

Contains the colour (in yuv) used to fill the screen when the video is
running in a window.
--------------------------------------------------------------------------------
2898
bits 0:23
        Decoder video window colour
        Contains the colour (in yuv) used to fill the video window when the
        video is turned off.
```

```
bit 24
        Decoder video output
        0 = Video on
        1 = Video off

bit 28
        Decoder plane order
        0 = Y,UV
        1 = UV,Y

bit 29
        Decoder second plane byte order
        0 = Normal (UV)
        1 = Swapped (VU)

In normal usage, the first plane is Y & the second plane is UV. Though the
order of the planes can be swapped, only the byte order of the second plane
can be swapped. This isn't much use for the Y plane, but can be useful for
the UV plane.

-------------------------------------------------------------------------------
289C
bits 0:15
        Decoder vertical field offset 1

bits 16:31
        Decoder vertical field offset 2

Controls field output vertical alignment. The higher the number, the lower
the image on screen. Known starting values are 0x011E0017 (NTSC) &
0x01500017 (PAL)
-------------------------------------------------------------------------------
28A0
bits 0:15
        Decoder & osd width in pixels

bits 16:31
        Decoder & osd height in pixels

All output from the decoder & osd are disabled beyond this area. Decoder
output will simply go black outside of this region. If the osd tries to
exceed this area it will become corrupt.
-------------------------------------------------------------------------------
28A4
bits 0:11
        osd left shift.

Has a range of 0x770->0x7FF. With the exception of 0, any value outside of
this range corrupts the osd.
-------------------------------------------------------------------------------
28A8
bits 0:15
        osd vertical field offset 1

bits 16:31
        osd vertical field offset 2

Controls field output vertical alignment. The higher the number, the lower
the image on screen. Known starting values are 0x011E0017 (NTSC) &
0x01500017 (PAL)
-------------------------------------------------------------------------------
28AC  --------    ?? unknown
|
```

```
V
28BC  --------    ?? unknown
------------------------------------------------------------------------------
28C0
bit 0
        Current output field
        0 = first field
        1 = second field

bits 16:31
        Current scanline
        The scanline counts from the top line of the first field
        through to the last line of the second field.
------------------------------------------------------------------------------
28C4  --------    ?? unknown
|
V
28F8  --------    ?? unknown
------------------------------------------------------------------------------
28FC
bit 0
        ?? unknown
        0 = Normal
        1 = Breaks decoder & osd output
------------------------------------------------------------------------------
2900
bits 0:31
        Decoder vertical Y alias register 1
---------------
2904
bits 0:31
        Decoder vertical Y alias register 2
---------------
2908
bits 0:31
        Decoder vertical Y alias trigger

These three registers control the vertical aliasing filter for the Y plane.
Operation is similar to the horizontal Y filter (2804). The only real
difference is that there are only two registers to set before accessing
the trigger register (2908). As for the horizontal filter, the values are
taken from a lookup table in the firmware, and the procedure must be
repeated 16 times to fully program the filter.
------------------------------------------------------------------------------
290C
bits 0:31
        Decoder vertical UV alias register 1
---------------
2910
bits 0:31
        Decoder vertical UV alias register 2
---------------
2914
bits 0:31
        Decoder vertical UV alias trigger

These three registers control the vertical aliasing filter for the UV
plane. Operation is the same as the Y filter, with 2914 being the trigger.
------------------------------------------------------------------------------
2918
bits 0:15
        Decoder Y source height in pixels
```

```
bits 16:31
        Decoder Y destination height in pixels
---------------
291C
bits 0:15
        Decoder UV source height in pixels divided by 2

bits 16:31
        Decoder UV destination height in pixels

NOTE: For both registers, the resulting image must be fully visible on
screen. If the image exceeds the bottom edge both the source and
destination size must be adjusted to reflect the visible portion. For the
source height, you must take into account the scaling when calculating the
new value.
--------------------------------------------------------------------------------
2920
bits 0:31
        Decoder Y vertical scaling
        Normally = Reg 2930 >> 2
---------------
2924
bits 0:31
        Decoder Y vertical scaling
        Normally = Reg 2920 + 0x514
---------------
2928
bits 0:31
        Decoder UV vertical scaling
        When enlarging = Reg 2930 >> 2
        When reducing = Reg 2930 >> 3
---------------
292C
bits 0:31
        Decoder UV vertical scaling
        Normally = Reg 2928 + 0x514
---------------
2930
bits 0:31
        Decoder 'master' value for vertical scaling
---------------
2934
bits 0:31
        Decoder ?? unknown - Y vertical scaling
---------------
2938
bits 0:31
        Decoder Y vertical scaling
        Normally = Reg 2930
---------------
293C
bits 0:31
        Decoder ?? unknown - Y vertical scaling
---------------
2940
bits 0:31
        Decoder UV vertical scaling
        When enlarging = Reg 2930 >> 1
        When reducing = Reg 2930
---------------
2944
bits 0:31
        Decoder ?? unknown - UV vertical scaling
```

```
---------------
2948
bits 0:31
        Decoder UV vertical scaling
        Normally = Reg 2940
---------------
294C
bits 0:31
        Decoder ?? unknown - UV vertical scaling

Most of these registers either control vertical scaling, or appear linked
to it in some way. Register 2930 contains the 'master' value & all other
registers can be calculated from that one. You must also remember to
correctly set the divider in Reg 296C

To enlarge:
        Reg 2930 = (source_height * 0x00200000) / destination_height
        Reg 296C = No divide

To reduce from full size down to half size:
        Reg 2930 = (source_height/2 * 0x00200000) / destination height
        Reg 296C = Divide by 2

To reduce from half down to quarter.
        Reg 2930 = (source_height/4 * 0x00200000) / destination height
        Reg 296C = Divide by 4

-------------------------------------------------------------------------------
2950
bits 0:15
        Decoder Y line index into display buffer, first field

bits 16:31
        Decoder Y vertical line skip, first field
-------------------------------------------------------------------------------
2954
bits 0:15
        Decoder Y line index into display buffer, second field

bits 16:31
        Decoder Y vertical line skip, second field
-------------------------------------------------------------------------------
2958
bits 0:15
        Decoder UV line index into display buffer, first field

bits 16:31
        Decoder UV vertical line skip, first field
-------------------------------------------------------------------------------
295C
bits 0:15
        Decoder UV line index into display buffer, second field

bits 16:31
        Decoder UV vertical line skip, second field
-------------------------------------------------------------------------------
2960
bits 0:15
        Decoder destination height minus 1

bits 16:31
        Decoder destination height divided by 2
-------------------------------------------------------------------------------
```

```
2964
bits 0:15
        Decoder Y vertical offset, second field

bits 16:31
        Decoder Y vertical offset, first field

These two registers shift the Y plane up. The higher the number, the
greater the shift.
--------------------------------------------------------------------------------
2968
bits 0:15
        Decoder UV vertical offset, second field

bits 16:31
        Decoder UV vertical offset, first field

These two registers shift the UV plane up. The higher the number, the
greater the shift.
--------------------------------------------------------------------------------
296C
bits 0:1
        Decoder vertical Y output size divider
        00 = No divide
        01 = Divide by 2
        10 = Divide by 4

bits 8:9
        Decoder vertical UV output size divider
        00 = No divide
        01 = Divide by 2
        10 = Divide by 4
--------------------------------------------------------------------------------
2970
bit 0
        Decoder ?? unknown
        0 = Normal
        1 = Affect video output levels

bit 16
        Decoder ?? unknown
        0 = Normal
        1 = Disable vertical filter

--------------------------------------------------------------------------------
2974  --------   ?? unknown
|
V
29EF  --------   ?? unknown
--------------------------------------------------------------------------------
2A00
bits 0:2
        osd colour mode
        000 = 8 bit indexed
        001 = 16 bit (565)
        010 = 15 bit (555)
        011 = 12 bit (444)
        100 = 32 bit (8888)

bits 4:5
        osd display bpp
        01 = 8 bit
        10 = 16 bit
```

```
        11 = 32 bit

bit 8
        osd global alpha
        0 = Off
        1 = On

bit 9
        osd local alpha
        0 = Off
        1 = On

bit 10
        osd colour key
        0 = Off
        1 = On

bit 11
        osd ?? unknown
        Must be 1

bit 13
        osd colour space
        0 = ARGB
        1 = AYVU

bits 16:31
        osd ?? unknown
        Must be 0x001B (some kind of buffer pointer ?)

When the bits-per-pixel is set to 8, the colour mode is ignored and
assumed to be 8 bit indexed. For 16 & 32 bits-per-pixel the colour depth
is honoured, and when using a colour depth that requires fewer bytes than
allocated the extra bytes are used as padding. So for a 32 bpp with 8 bit
index colour, there are 3 padding bytes per pixel. It's also possible to
select 16bpp with a 32 bit colour mode. This results in the pixel width
being doubled, but the color key will not work as expected in this mode.

Colour key is as it suggests. You designate a colour which will become
completely transparent. When using 565, 555 or 444 colour modes, the
colour key is always 16 bits wide. The colour to key on is set in Reg 2A18.

Local alpha works differently depending on the colour mode. For 32bpp & 8
bit indexed, local alpha is a per-pixel 256 step transparency, with 0 being
transparent and 255 being solid. For the 16bpp modes 555 & 444, the unused
bit(s) act as a simple transparency switch, with 0 being solid & 1 being
fully transparent. There is no local alpha support for 16bit 565.

Global alpha is a 256 step transparency that applies to the entire osd,
with 0 being transparent & 255 being solid.

It's possible to combine colour key, local alpha & global alpha.
--------------------------------------------------------------------------------
2A04
bits 0:15
        osd x coord for left edge

bits 16:31
        osd y coord for top edge
---------------
2A08
bits 0:15
        osd x coord for right edge
```

```
bits 16:31
        osd y coord for bottom edge

For both registers, (0,0) = top left corner of the display area. These
registers do not control the osd size, only where it's positioned & how
much is visible. The visible osd area cannot exceed the right edge of the
display, otherwise the osd will become corrupt. See reg 2A10 for
setting osd width.
-------------------------------------------------------------------------------
2A0C
bits 0:31
        osd buffer index

An index into the osd buffer. Slowly incrementing this moves the osd left,
wrapping around onto the right edge
-------------------------------------------------------------------------------
2A10
bits 0:11
        osd buffer 32 bit word width

Contains the width of the osd measured in 32 bit words. This means that all
colour modes are restricted to a byte width which is divisible by 4.
-------------------------------------------------------------------------------
2A14
bits 0:15
        osd height in pixels

bits 16:32
        osd line index into buffer
        osd will start displaying from this line.
-------------------------------------------------------------------------------
2A18
bits 0:31
        osd colour key

Contains the colour value which will be transparent.
-------------------------------------------------------------------------------
2A1C
bits 0:7
        osd global alpha

Contains the global alpha value (equiv ivtvfbctl --alpha XX)
-------------------------------------------------------------------------------
2A20  --------    ?? unknown
|
V
2A2C  --------    ?? unknown
-------------------------------------------------------------------------------
2A30
bits 0:7
        osd colour to change in indexed palette
---------------
2A34
bits 0:31
        osd colour for indexed palette

To set the new palette, first load the index of the colour to change into
2A30, then load the new colour into 2A34. The full palette is 256 colours,
so the index range is 0x00-0xFF
-------------------------------------------------------------------------------
2A38  --------    ?? unknown
2A3C  --------    ?? unknown
```

```
------------------------------------------------------------------------------
2A40
bits 0:31
        osd ?? unknown

Affects overall brightness, wrapping around to black
------------------------------------------------------------------------------
2A44
bits 0:31
        osd ?? unknown

Green tint
------------------------------------------------------------------------------
2A48
bits 0:31
        osd ?? unknown

Red tint
------------------------------------------------------------------------------
2A4C
bits 0:31
        osd ?? unknown

Affects overall brightness, wrapping around to black
------------------------------------------------------------------------------
2A50
bits 0:31
        osd ?? unknown

Colour shift
------------------------------------------------------------------------------
2A54
bits 0:31
        osd ?? unknown

Colour shift
------------------------------------------------------------------------------
2A58  --------    ?? unknown
|
V
2AFC  --------    ?? unknown
------------------------------------------------------------------------------
2B00
bit 0
        osd filter control
        0 = filter off
        1 = filter on

bits 1:4
        osd ?? unknown

------------------------------------------------------------------------------
```

4.11.9 The cx231xx DMA engine

This page describes the structures and procedures used by the cx2341x DMA engine.

Introduction

The cx2341x PCI interface is busmaster capable. This means it has a DMA engine to efficiently transfer large volumes of data between the card and main memory without requiring help from a CPU. Like most hardware, it must operate on contiguous physical memory. This is difficult to come by in large quantities on virtual memory machines.

Therefore, it also supports a technique called “scatter-gather”. The card can transfer multiple buffers in one operation. Instead of allocating one large contiguous buffer, the driver can allocate several smaller buffers.

In practice, I’ve seen the average transfer to be roughly 80K, but transfers above 128K were not uncommon, particularly at startup. The 128K figure is important, because that is the largest block that the kernel can normally allocate. Even still, 128K blocks are hard to come by, so the driver writer is urged to choose a smaller block size and learn the scatter-gather technique.

Mailbox #10 is reserved for DMA transfer information.

Note: the hardware expects little-endian data (‘intel format’).

Flow

This section describes, in general, the order of events when handling DMA transfers. Detailed information follows this section.

- The card raises the Encoder interrupt.
- The driver reads the transfer type, offset and size from Mailbox #10.
- The driver constructs the scatter-gather array from enough free dma buffers to cover the size.
- The driver schedules the DMA transfer via the ScheduleDMAtoHost API call.
- The card raises the DMA Complete interrupt.
- The driver checks the DMA status register for any errors.
- The driver post-processes the newly transferred buffers.

NOTE! It is possible that the Encoder and DMA Complete interrupts get raised simultaneously. (End of the last, start of the next, etc.)

Mailbox #10

The Flags, Command, Return Value and Timeout fields are ignored.

- Name: Mailbox #10
- Results[0]: Type: 0: MPEG.
- Results[1]: Offset: The position relative to the card’s memory space.
- Results[2]: Size: The exact number of bytes to transfer.

My speculation is that since the StartCapture API has a capture type of “RAW” available, that the type field will have other values that correspond to YUV and PCM data.

Scatter-Gather Array

The scatter-gather array is a contiguously allocated block of memory that tells the card the source and destination of each data-block to transfer. Card “addresses” are derived from the offset supplied by Mailbox #10. Host addresses are the physical memory location of the target DMA buffer.

Each S-G array element is a struct of three 32-bit words. The first word is the source address, the second is the destination address. Both take up the entire 32 bits. The lowest 18 bits of the third word is the

transfer byte count. The high-bit of the third word is the “last” flag. The last-flag tells the card to raise the DMA_DONE interrupt. From hard personal experience, if you forget to set this bit, the card will still “work” but the stream will most likely get corrupted.

The transfer count must be a multiple of 256. Therefore, the driver will need to track how much data in the target buffer is valid and deal with it accordingly.

Array Element:

- 32-bit Source Address
- 32-bit Destination Address
- 14-bit reserved (high bit is the last flag)
- 18-bit byte count

DMA Transfer Status

Register 0x0004 holds the DMA Transfer Status:

- bit 0: read completed
- bit 1: write completed
- bit 2: DMA read error
- bit 3: DMA write error
- bit 4: Scatter-Gather array error

4.11.10 Non-compressed file format

The cx23416 can produce (and the cx23415 can also read) raw YUV output. The format of a YUV frame is specific to this chip and is called HM12. ‘HM’ stands for ‘Hauppauge Macroblock’, which is a misnomer as ‘Conexant Macroblock’ would be more accurate.

The format is YUV 4:2:0 which uses 1 Y byte per pixel and 1 U and V byte per four pixels.

The data is encoded as two macroblock planes, the first containing the Y values, the second containing UV macroblocks.

The Y plane is divided into blocks of 16x16 pixels from left to right and from top to bottom. Each block is transmitted in turn, line-by-line.

So the first 16 bytes are the first line of the top-left block, the second 16 bytes are the second line of the top-left block, etc. After transmitting this block the first line of the block on the right to the first block is transmitted, etc.

The UV plane is divided into blocks of 16x8 UV values going from left to right, top to bottom. Each block is transmitted in turn, line-by-line.

So the first 16 bytes are the first line of the top-left block and contain 8 UV value pairs (16 bytes in total). The second 16 bytes are the second line of 8 UV pairs of the top-left block, etc. After transmitting this block the first line of the block on the right to the first block is transmitted, etc.

The code below is given as an example on how to convert HM12 to separate Y, U and V planes. This code assumes frames of 720x576 (PAL) pixels.

The width of a frame is always 720 pixels, regardless of the actual specified width.

If the height is not a multiple of 32 lines, then the captured video is missing macroblocks at the end and is unusable. So the height must be a multiple of 32.

Raw format c example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static unsigned char frame[576*720*3/2];
static unsigned char framey[576*720];
static unsigned char frameu[576*720 / 4];
static unsigned char framev[576*720 / 4];

static void de_macro_y(unsigned char* dst, unsigned char *src, int dstride, int w, int h)
{
unsigned int y, x, i;

// descramble Y plane
// dstride = 720 = w
// The Y plane is divided into blocks of 16x16 pixels
// Each block in transmitted in turn, line-by-line.
for (y = 0; y < h; y += 16) {
        for (x = 0; x < w; x += 16) {
        for (i = 0; i < 16; i++) {
                memcpy(dst + x + (y + i) * dstride, src, 16);
                src += 16;
        }
        }
}
}

static void de_macro_uv(unsigned char *dstu, unsigned char *dstv, unsigned char *src, int␣
 ↪dstride, int w, int h)
{
unsigned int y, x, i;

// descramble U/V plane
// dstride = 720 / 2 = w
// The U/V values are interlaced (UVUV...).
// Again, the UV plane is divided into blocks of 16x16 UV values.
// Each block in transmitted in turn, line-by-line.
for (y = 0; y < h; y += 16) {
        for (x = 0; x < w; x += 8) {
        for (i = 0; i < 16; i++) {
                int idx = x + (y + i) * dstride;

                dstu[idx+0] = src[0];  dstv[idx+0] = src[1];
                dstu[idx+1] = src[2];  dstv[idx+1] = src[3];
                dstu[idx+2] = src[4];  dstv[idx+2] = src[5];
                dstu[idx+3] = src[6];  dstv[idx+3] = src[7];
                dstu[idx+4] = src[8];  dstv[idx+4] = src[9];
                dstu[idx+5] = src[10]; dstv[idx+5] = src[11];
                dstu[idx+6] = src[12]; dstv[idx+6] = src[13];
                dstu[idx+7] = src[14]; dstv[idx+7] = src[15];
                src += 16;
        }
        }
}
}

/*************************************************************************/
int main(int argc, char **argv)
{
FILE *fin;
int i;
```

```
if (argc == 1) fin = stdin;
else fin = fopen(argv[1], "r");

if (fin == NULL) {
        fprintf(stderr, "cannot open input\n");
        exit(-1);
}
while (fread(frame, sizeof(frame), 1, fin) == 1) {
        de_macro_y(framey, frame, 720, 720, 576);
        de_macro_uv(frameu, framev, frame + 720 * 576, 720 / 2, 720 / 2, 576 / 2);
        fwrite(framey, sizeof(framey), 1, stdout);
        fwrite(framev, sizeof(framev), 1, stdout);
        fwrite(frameu, sizeof(frameu), 1, stdout);
}
fclose(fin);
return 0;
}
```

4.11.11 Format of embedded V4L2_MPEG_STREAM_VBI_FMT_IVTV VBI data

Author: Hans Verkuil <hverkuil@xs4all.nl>

This section describes the V4L2_MPEG_STREAM_VBI_FMT_IVTV format of the VBI data embedded in an MPEG-2 program stream. This format is in part dictated by some hardware limitations of the ivtv driver (the driver for the Conexant cx23415/6 chips), in particular a maximum size for the VBI data. Anything longer is cut off when the MPEG stream is played back through the cx23415.

The advantage of this format is it is very compact and that all VBI data for all lines can be stored while still fitting within the maximum allowed size.

The stream ID of the VBI data is 0xBD. The maximum size of the embedded data is 4 + 43 * 36, which is 4 bytes for a header and 2 * 18 VBI lines with a 1 byte header and a 42 bytes payload each. Anything beyond this limit is cut off by the cx23415/6 firmware. Besides the data for the VBI lines we also need 36 bits for a bitmask determining which lines are captured and 4 bytes for a magic cookie, signifying that this data package contains V4L2_MPEG_STREAM_VBI_FMT_IVTV VBI data. If all lines are used, then there is no longer room for the bitmask. To solve this two different magic numbers were introduced:

'itv0': After this magic number two unsigned longs follow. Bits 0-17 of the first unsigned long denote which lines of the first field are captured. Bits 18-31 of the first unsigned long and bits 0-3 of the second unsigned long are used for the second field.

'ITV0': This magic number assumes all VBI lines are captured, i.e. it implicitly implies that the bitmasks are 0xffffffff and 0xf.

After these magic cookies (and the 8 byte bitmask in case of cookie 'itv0') the captured VBI lines start:

For each line the least significant 4 bits of the first byte contain the data type. Possible values are shown in the table below. The payload is in the following 42 bytes.

Here is the list of possible data types:

```
#define IVTV_SLICED_TYPE_TELETEXT       0x1     // Teletext (uses lines 6-22 for PAL)
#define IVTV_SLICED_TYPE_CC             0x4     // Closed Captions (line 21 NTSC)
#define IVTV_SLICED_TYPE_WSS            0x5     // Wide Screen Signal (line 23 PAL)
#define IVTV_SLICED_TYPE_VPS            0x7     // Video Programming System (PAL) (line 16)
```

4.12 The cx88 driver

Author: Gerd Hoffmann

This is a v4l2 device driver for the cx2388x chip.

4.12.1 Current status

video

- Works.
- Overlay isn't supported.

audio

- Works. The TV standard detection is made by the driver, as the hardware has bugs to autodetect.
- audio data dma (i.e. recording without loopback cable to the sound card) is supported via cx88-alsa.

vbi

- Works.

4.12.2 How to add support for new cards

The driver needs some config info for the TV cards. This stuff is in cx88-cards.c. If the driver doesn't work well you likely need a new entry for your card in that file. Check the kernel log (using dmesg) to see whenever the driver knows your card or not. There is a line like this one:

```
cx8800[0]: subsystem: 0070:3400, board: Hauppauge WinTV \
        34xxx models [card=1,autodetected]
```

If your card is listed as "board: UNKNOWN/GENERIC" it is unknown to the driver. What to do then?

1. Try upgrading to the latest snapshot, maybe it has been added meanwhile.
2. You can try to create a new entry yourself, have a look at cx88-cards.c. If that worked, mail me your changes as unified diff ("diff -u").
3. Or you can mail me the config information. We need at least the following information to add the card:
   - the PCI Subsystem ID ("0070:3400" from the line above, "lspci -v" output is fine too).
   - the tuner type used by the card. You can try to find one by trial-and-error using the tuner=<n> insmod option. If you know which one the card has you can also have a look at the list in CARDLIST.tuner

4.12.3 Documentation missing at the cx88 datasheet

MO_OUTPUT_FORMAT (0x310164)

```
Previous default from DScaler: 0x1c1f0008
Digit 8: 31-28
28: PREVREMOD = 1

Digit 7: 27-24 (0xc = 12 = b1100 )
27: COMBALT = 1
26: PAL_INV_PHASE
  (DScaler apparently set this to 1, resulted in sucky picture)

Digits 6,5: 23-16
25-16: COMB_RANGE = 0x1f [default] (9 bits -> max 512)
```

```
Digit 4: 15-12
15: DISIFX = 0
14: INVCBF = 0
13: DISADAPT = 0
12: NARROWADAPT = 0

Digit 3: 11-8
11: FORCE2H
10: FORCEREMD
9: NCHROMAEN
8: NREMODEN

Digit 2: 7-4
7-6: YCORE
5-4: CCORE

Digit 1: 3-0
3: RANGE = 1
2: HACTEXT
1: HSFMT
```

0x47 is the sync byte for MPEG-2 transport stream packets. Datasheet incorrectly states to use 47 decimal. 188 is the length. All DVB compliant frontends output packets with this start code.

4.12.4 Hauppauge WinTV cx88 IR information

The controls for the mux are GPIO [0,1] for source, and GPIO 2 for muting.

GPIO0	GPIO1	
0	0	TV Audio
1	0	FM radio
0	1	Line-In
1	1	Mono tuner bypass or CD passthru (tuner specific)

GPIO 16(I believe) is tied to the IR port (if present).

From the data sheet:

- Register 24'h20004 PCI Interrupt Status
- bit [18] IR_SMP_INT Set when 32 input samples have been collected over
- gpio[16] pin into GP_SAMPLE register.

What's missing from the data sheet:

- Setup 4KHz sampling rate (roughly 2x oversampled; good enough for our RC5 compat remote)
- set register 0x35C050 to 0xa80a80
- enable sampling
- set register 0x35C054 to 0x5
- enable the IRQ bit 18 in the interrupt mask register (and provide for a handler)

GP_SAMPLE register is at 0x35C058

Bits are then right shifted into the GP_SAMPLE register at the specified rate; you get an interrupt when a full DWORD is received. You need to recover the actual RC5 bits out of the (oversampled) IR sensor bits. (Hint: look for the 0/1and 1/0 crossings of the RC5 bi-phase data) An actual raw RC5 code will span 2-3 DWORDS, depending on the actual alignment.

I'm pretty sure when no IR signal is present the receiver is always in a marking state(1); but stray light, etc can cause intermittent noise values as well. Remember, this is a free running sample of the IR receiver state over time, so don't assume any sample starts at any particular place.

Additional info

This data sheet (google search) seems to have a lovely description of the RC5 basics: http://www.atmel.com/dyn/resources/prod_documents/doc2817.pdf

This document has more data: http://www.nenya.be/beor/electronics/rc5.htm

This document has a how to decode a bi-phase data stream: http://www.ee.washington.edu/circuit_archive/text/ir_decode.txt

This document has still more info: http://www.xs4all.nl/~sbp/knowledge/ir/rc5.htm

4.13 The VPBE V4L2 driver design

4.13.1 File partitioning

V4L2 display device driver drivers/media/platform/davinci/vpbe_display.c drivers/media/platform/davinci/vpbe_display.h

VPBE display controller drivers/media/platform/davinci/vpbe.c drivers/media/platform/davinci/vpbe.h

VPBE venc sub device driver drivers/media/platform/davinci/vpbe_venc.c drivers/media/platform/davinci/vpbe_venc.h drivers/media/platform/davinci/vpbe_venc_regs.h

VPBE osd driver drivers/media/platform/davinci/vpbe_osd.c drivers/media/platform/davinci/vpbe_osd.h drivers/media/platform/davinci/vpbe_osd_regs.h

4.13.2 Functional partitioning

Consists of the following (in the same order as the list under file partitioning):

1. V4L2 display driver Implements creation of video2 and video3 device nodes and provides v4l2 device interface to manage VID0 and VID1 layers.
2. Display controller Loads up VENC, OSD and external encoders such as ths8200. It provides a set of API calls to V4L2 drivers to set the output/standards in the VENC or external sub devices. It also provides a device object to access the services from OSD subdevice using sub device ops. The connection of external encoders to VENC LCD controller port is done at init time based on default output and standard selection or at run time when application change the output through V4L2 IOCTLs.

   When connected to an external encoder, vpbe controller is also responsible for setting up the interface between VENC and external encoders based on board specific settings (specified in board-xxx-evm.c). This allows interfacing external encoders such as ths8200. The setup_if_config() is implemented for this as well as configure_venc() (part of the next patch) API to set timings in VENC for a specific display resolution. As of this patch series, the interconnection and enabling and setting of the external encoders is not present, and would be a part of the next patch series.
3. VENC subdevice module Responsible for setting outputs provided through internal DACs and also setting timings at LCD controller port when external encoders are connected at the port or LCD panel timings required. When external encoder/LCD panel is connected, the timings for a specific standard/preset is retrieved from the board specific table and the values are used to set the timings in venc using non-standard timing mode.

   Support LCD Panel displays using the VENC. For example to support a Logic PD display, it requires setting up the LCD controller port with a set of timings for the resolution supported and setting the dot clock. So we could add the available outputs as a board specific entry (i.e add the “LogicPD” output name to board-xxx-evm.c). A table of timings for various LCDs supported can be maintained in the board specific setup file to support various LCD displays.As of this patch a basic driver is present, and this support for external encoders and displays forms a part of the next patch series.

4. OSD module OSD module implements all OSD layer management and hardware specific features. The VPBE module interacts with the OSD for enabling and disabling appropriate features of the OSD.

4.13.3 Current status

A fully functional working version of the V4L2 driver is available. This driver has been tested with NTSC and PAL standards and buffer streaming.

4.13.4 To be done

vpbe display controller

- Add support for external encoders.
- add support for selecting external encoder as default at probe time.

vpbe venc sub device

- add timings for supporting ths8200
- add support for LogicPD LCD.

FB drivers

- Add support for fbdev drivers.- Ready and part of subsequent patches.

4.14 The Samsung S5P/EXYNOS4 FIMC driver

The FIMC (Fully Interactive Mobile Camera) device available in Samsung SoC Application Processors is an integrated camera host interface, color space converter, image resizer and rotator. It's also capable of capturing data from LCD controller (FIMD) through the SoC internal writeback data path. There are multiple FIMC instances in the SoCs (up to 4), having slightly different capabilities, like pixel alignment constraints, rotator availability, LCD writeback support, etc. The driver is located at drivers/media/platform/exynos4-is directory.

4.14.1 Supported SoCs

S5PC100 (mem-to-mem only), S5PV210, EXYNOS4210

4.14.2 Supported features

- camera parallel interface capture (ITU-R.BT601/565);
- camera serial interface capture (MIPI-CSI2);
- memory-to-memory processing (color space conversion, scaling, mirror and rotation);
- dynamic pipeline re-configuration at runtime (re-attachment of any FIMC instance to any parallel video input or any MIPI-CSI front-end);
- runtime PM and system wide suspend/resume

4.14.3 Not currently supported

- LCD writeback input
- per frame clock gating (mem-to-mem)

4.14.4 Files partitioning

- media device driver drivers/media/platform/exynos4-is/media-dev.[ch]
- camera capture video device driver drivers/media/platform/exynos4-is/fimc-capture.c
- MIPI-CSI2 receiver subdev drivers/media/platform/exynos4-is/mipi-csis.[ch]
- video post-processor (mem-to-mem) drivers/media/platform/exynos4-is/fimc-core.c
- common files drivers/media/platform/exynos4-is/fimc-core.h drivers/media/platform/exynos4-is/fimc-reg.h drivers/media/platform/exynos4-is/regs-fimc.h

4.14.5 User space interfaces

Media device interface

The driver supports Media Controller API as defined at *Part IV - Media Controller API* . The media device driver name is “SAMSUNG S5P FIMC”.

The purpose of this interface is to allow changing assignment of FIMC instances to the SoC peripheral camera input at runtime and optionally to control internal connections of the MIPI-CSIS device(s) to the FIMC entities.

The media device interface allows to configure the SoC for capturing image data from the sensor through more than one FIMC instance (e.g. for simultaneous viewfinder and still capture setup). Reconfiguration is done by enabling/disabling media links created by the driver during initialization. The internal device topology can be easily discovered through media entity and links enumeration.

Memory-to-memory video node

V4L2 memory-to-memory interface at /dev/video? device node. This is standalone video device, it has no media pads. However please note the mem-to-mem and capture video node operation on same FIMC instance is not allowed. The driver detects such cases but the applications should prevent them to avoid an undefined behaviour.

Capture video node

The driver supports V4L2 Video Capture Interface as defined at *Interfaces* .

At the capture and mem-to-mem video nodes only the multi-planar API is supported. For more details see: *Single- and multi-planar APIs* .

Camera capture subdevs

Each FIMC instance exports a sub-device node (/dev/v4l-subdev?), a sub-device node is also created per each available and enabled at the platform level MIPI-CSI receiver device (currently up to two).

sysfs

In order to enable more precise camera pipeline control through the sub-device API the driver creates a sysfs entry associated with “s5p-fimc-md” platform device. The entry path is: /sys/platform/devices/s5p-fimc-md/subdev_conf_mode.

In typical use case there could be a following capture pipeline configuration: sensor subdev -> mipi-csi subdev -> fimc subdev -> video node

When we configure these devices through sub-device API at user space, the configuration flow must be from left to right, and the video node is configured as last one. When we don’t use sub-device user

space API the whole configuration of all devices belonging to the pipeline is done at the video node driver. The sysfs entry allows to instruct the capture node driver not to configure the sub-devices (format, crop), to avoid resetting the subdevs' configuration when the last configuration steps at the video node is performed.

For full sub-device control support (subdevs configured at user space before starting streaming):

```
# echo "sub-dev" > /sys/platform/devices/s5p-fimc-md/subdev_conf_mode
```

For V4L2 video node control only (subdevs configured internally by the host driver):

```
# echo "vid-dev" > /sys/platform/devices/s5p-fimc-md/subdev_conf_mode
```

This is a default option.

4.14.6 5. Device mapping to video and subdev device nodes

There are associated two video device nodes with each device instance in hardware - video capture and mem-to-mem and additionally a subdev node for more precise FIMC capture subsystem control. In addition a separate v4l2 sub-device node is created per each MIPI-CSIS device.

How to find out which /dev/video? or /dev/v4l-subdev? is assigned to which device?

You can either grep through the kernel log to find relevant information, i.e.

```
# dmesg | grep -i fimc
```

(note that udev, if present, might still have rearranged the video nodes),

or retrieve the information from /dev/media? with help of the media-ctl tool:

```
# media-ctl -p
```

4.14.7 7. Build

If the driver is built as a loadable kernel module (CONFIG_VIDEO_SAMSUNG_S5P_FIMC=m) two modules are created (in addition to the core v4l2 modules): s5p-fimc.ko and optional s5p-csis.ko (MIPI-CSI receiver subdev).

4.15 The ivtv driver

Author: Hans Verkuil <hverkuil@xs4all.nl>

This is a v4l2 device driver for the Conexant cx23415/6 MPEG encoder/decoder. The cx23415 can do both encoding and decoding, the cx23416 can only do MPEG encoding. Currently the only card featuring full decoding support is the Hauppauge PVR-350.

Note:

1. *This driver requires the latest encoder firmware (version 2.06.039, size 376836 bytes). Get the firmware from here:*
   https://linuxtv.org/downloads/firmware/#conexant
2. *'normal' TV applications do not work with this driver, you need an application that can handle MPEG input such as mplayer, xine, MythTV, etc.*

The primary goal of the IVTV project is to provide a "clean room" Linux Open Source driver implementation for video capture cards based on the iCompression iTVC15 or Conexant CX23415/CX23416 MPEG Codec.

4.15.1 Features

- Hardware mpeg2 capture of broadcast video (and sound) via the tuner or S-Video/Composite and audio line-in.
- Hardware mpeg2 capture of FM radio where hardware support exists
- Supports NTSC, PAL, SECAM with stereo sound
- Supports SAP and bilingual transmissions.
- Supports raw VBI (closed captions and teletext).
- Supports sliced VBI (closed captions and teletext) and is able to insert this into the captured MPEG stream.
- Supports raw YUV and PCM input.

4.15.2 Additional features for the PVR-350 (CX23415 based)

- Provides hardware mpeg2 playback
- Provides comprehensive OSD (On Screen Display: ie. graphics overlaying the video signal)
- Provides a framebuffer (allowing X applications to appear on the video device)
- Supports raw YUV output.

IMPORTANT: In case of problems first read this page: https://help.ubuntu.com/community/Install_IVTV_Troubleshooting

4.15.3 See also

https://linuxtv.org

4.15.4 IRC

irc://irc.freenode.net/#v4l

4.15.5 Devices

A maximum of 12 ivtv boards are allowed at the moment.

Cards that don't have a video output capability (i.e. non PVR350 cards) lack the vbi8, vbi16, video16 and video48 devices. They also do not support the framebuffer device /dev/fbx for OSD.

The radio0 device may or may not be present, depending on whether the card has a radio tuner or not.

Here is a list of the base v4l devices:

```
crw-rw----    1 root     video     81,   0 Jun 19 22:22 /dev/video0
crw-rw----    1 root     video     81,  16 Jun 19 22:22 /dev/video16
crw-rw----    1 root     video     81,  24 Jun 19 22:22 /dev/video24
crw-rw----    1 root     video     81,  32 Jun 19 22:22 /dev/video32
crw-rw----    1 root     video     81,  48 Jun 19 22:22 /dev/video48
crw-rw----    1 root     video     81,  64 Jun 19 22:22 /dev/radio0
crw-rw----    1 root     video     81, 224 Jun 19 22:22 /dev/vbi0
crw-rw----    1 root     video     81, 228 Jun 19 22:22 /dev/vbi8
crw-rw----    1 root     video     81, 232 Jun 19 22:22 /dev/vbi16
```

4.15.6 Base devices

For every extra card you have the numbers increased by one. For example, /dev/video0 is listed as the 'base' encoding capture device so we have:

- /dev/video0 is the encoding capture device for the first card (card 0)
- /dev/video1 is the encoding capture device for the second card (card 1)
- /dev/video2 is the encoding capture device for the third card (card 2)

Note that if the first card doesn't have a feature (eg no decoder, so no video16, the second card will still use video17. The simple rule is 'add the card number to the base device number'. If you have other capture cards (e.g. WinTV PCI) that are detected first, then you have to tell the ivtv module about it so that it will start counting at 1 (or 2, or whatever). Otherwise the device numbers can get confusing. The ivtv 'ivtv_first_minor' module option can be used for that.

- /dev/video0

  The encoding capture device(s).

  Read-only.

  Reading from this device gets you the MPEG1/2 program stream. Example:

```
cat /dev/video0 > my.mpg (you need to hit ctrl-c to exit)
```

- /dev/video16

  The decoder output device(s)

  Write-only. Only present if the MPEG decoder (i.e. CX23415) exists.

  An mpeg2 stream sent to this device will appear on the selected video display, audio will appear on the line-out/audio out. It is only available for cards that support video out. Example:

```
cat my.mpg >/dev/video16
```

- /dev/video24

  The raw audio capture device(s).

  Read-only

  The raw audio PCM stereo stream from the currently selected tuner or audio line-in. Reading from this device results in a raw (signed 16 bit Little Endian, 48000 Hz, stereo pcm) capture. This device only captures audio. This should be replaced by an ALSA device in the future. Note that there is no corresponding raw audio output device, this is not supported in the decoder firmware.

- /dev/video32

  The raw video capture device(s)

  Read-only

  The raw YUV video output from the current video input. The YUV format is non-standard (V4L2_PIX_FMT_HM12).

  Note that the YUV and PCM streams are not synchronized, so they are of limited use.

- /dev/video48

  The raw video display device(s)

  Write-only. Only present if the MPEG decoder (i.e. CX23415) exists.

  Writes a YUV stream to the decoder of the card.

- /dev/radio0

  The radio tuner device(s)

  Cannot be read or written.

  Used to enable the radio tuner and tune to a frequency. You cannot read or write audio streams with this device. Once you use this device to tune the radio, use /dev/video24 to read the raw pcm stream or /dev/video0 to get an mpeg2 stream with black video.

- /dev/vbi0

  The 'vertical blank interval' (Teletext, CC, WSS etc) capture device(s)

  Read-only

  Captures the raw (or sliced) video data sent during the Vertical Blank Interval. This data is used to encode teletext, closed captions, VPS, widescreen signalling, electronic program guide information, and other services.

- /dev/vbi8

  Processed vbi feedback device(s)

  Read-only. Only present if the MPEG decoder (i.e. CX23415) exists.

  The sliced VBI data embedded in an MPEG stream is reproduced on this device. So while playing back a recording on /dev/video16, you can read the embedded VBI data from /dev/vbi8.

- /dev/vbi16

  The vbi 'display' device(s)

  Write-only. Only present if the MPEG decoder (i.e. CX23415) exists.

  Can be used to send sliced VBI data to the video-out connector.

4.16 Vaio Picturebook Motion Eye Camera Driver

This driver enable the use of video4linux compatible applications with the Motion Eye camera. This driver requires the "Sony Laptop Extras" driver (which can be found in the "Misc devices" section of the kernel configuration utility) to be compiled and installed (using its "camera=1" parameter).

It can do at maximum 30 fps @ 320x240 or 15 fps @ 640x480.

Grabbing is supported in packed YUV colorspace only.

MJPEG hardware grabbing is supported via a private API (see below).

4.16.1 Hardware supported

This driver supports the 'second' version of the MotionEye camera :)

The first version was connected directly on the video bus of the Neomagic video card and is unsupported.

The second one, made by Kawasaki Steel is fully supported by this driver (PCI vendor/device is 0x136b/0xff01)

The third one, present in recent (more or less last year) Picturebooks (C1M* models), is not supported. The manufacturer has given the specs to the developers under a NDA (which allows the development

of a GPL driver however), but things are not moving very fast (see http://r-engine.sourceforge.net/) (PCI vendor/device is 0x10cf/0x2011).

There is a forth model connected on the USB bus in TR1* Vaio laptops. This camera is not supported at all by the current driver, in fact little information if any is available for this camera (USB vendor/device is 0x054c/0x0107).

4.16.2 Driver options

Several options can be passed to the meye driver using the standard module argument syntax (<param>=<value> when passing the option to the module or meye.<param>=<value> on the kernel boot line when meye is statically linked into the kernel). Those options are:

```
gbuffers:       number of capture buffers, default is 2 (32 max)

gbufsize:       size of each capture buffer, default is 614400

video_nr:       video device to register (0 = /dev/video0, etc)
```

4.16.3 Module use

In order to automatically load the meye module on use, you can put those lines in your /etc/modprobe.d/meye.conf file:

```
alias char-major-81 videodev
alias char-major-81-0 meye
options meye gbuffers=32
```

4.16.4 Usage:

```
xawtv >= 3.49 (<http://bytesex.org/xawtv/>)
        for display and uncompressed video capture:

                xawtv -c /dev/video0 -geometry 640x480
                        or
                xawtv -c /dev/video0 -geometry 320x240

motioneye (<http://popies.net/meye/>)
        for getting ppm or jpg snapshots, mjpeg video
```

4.16.5 Private API

The driver supports frame grabbing with the video4linux API, so all video4linux tools (like xawtv) should work with this driver.

Besides the video4linux interface, the driver has a private interface for accessing the Motion Eye extended parameters (camera sharpness, agc, video framerate), the shapshot and the MJPEG capture facilities.

This interface consists of several ioctls (prototypes and structures can be found in include/linux/meye.h):

MEYEIOC_G_PARAMS and MEYEIOC_S_PARAMS Get and set the extended parameters of the motion eye camera. The user should always query the current parameters with MEYEIOC_G_PARAMS, change what he likes and then issue the MEYEIOC_S_PARAMS call (checking for -EINVAL). The extended parameters are described by the meye_params structure.

MEYEIOC_QBUF_CAPT Queue a buffer for capture (the buffers must have been obtained with a VIDIOCGMBUF call and mmap'ed by the application). The argument to MEYEIOC_QBUF_CAPT is the buffer number to queue (or -1 to end capture). The first call to MEYEIOC_QBUF_CAPT starts the streaming capture.

MEYEIOC_SYNC Takes as an argument the buffer number you want to sync. This ioctl blocks until the buffer is filled and ready for the application to use. It returns the buffer size.

MEYEIOC_STILLCAPT and MEYEIOC_STILLJCAPT Takes a snapshot in an uncompressed or compressed jpeg format. This ioctl blocks until the snapshot is done and returns (for jpeg snapshot) the size of the image. The image data is available from the first mmap'ed buffer.

Look at the 'motioneye' application code for an actual example.

4.16.6 Bugs / Todo

- 'motioneye' still uses the meye private v4l1 API extensions.

4.17 OMAP 3 Image Signal Processor (ISP) driver

Contacts: Laurent Pinchart <laurent.pinchart@ideasonboard.com>, Sakari Ailus <sakari.ailus@iki.fi>, David Cohen <dacohen@gmail.com>

4.17.1 Introduction

This file documents the Texas Instruments OMAP 3 Image Signal Processor (ISP) driver located under drivers/media/platform/omap3isp. The original driver was written by Texas Instruments but since that it has been rewritten (twice) at Nokia.

The driver has been successfully used on the following versions of OMAP 3:

- 3430
- 3530
- 3630

The driver implements V4L2, Media controller and v4l2_subdev interfaces. Sensor, lens and flash drivers using the v4l2_subdev interface in the kernel are supported.

4.17.2 Split to subdevs

The OMAP 3 ISP is split into V4L2 subdevs, each of the blocks inside the ISP having one subdev to represent it. Each of the subdevs provide a V4L2 subdev interface to userspace.

- OMAP3 ISP CCP2
- OMAP3 ISP CSI2a
- OMAP3 ISP CCDC
- OMAP3 ISP preview
- OMAP3 ISP resizer
- OMAP3 ISP AEWB
- OMAP3 ISP AF

- OMAP3 ISP histogram

Each possible link in the ISP is modelled by a link in the Media controller interface. For an example program see [2].

4.17.3 Controlling the OMAP 3 ISP

In general, the settings given to the OMAP 3 ISP take effect at the beginning of the following frame. This is done when the module becomes idle during the vertical blanking period on the sensor. In memory-to-memory operation the pipe is run one frame at a time. Applying the settings is done between the frames.

All the blocks in the ISP, excluding the CSI-2 and possibly the CCP2 receiver, insist on receiving complete frames. Sensors must thus never send the ISP partial frames.

Autoidle does have issues with some ISP blocks on the 3430, at least. Autoidle is only enabled on 3630 when the omap3isp module parameter autoidle is non-zero.

4.17.4 Events

The OMAP 3 ISP driver does support the V4L2 event interface on CCDC and statistics (AEWB, AF and histogram) subdevs.

The CCDC subdev produces V4L2_EVENT_FRAME_SYNC type event on HS_VS interrupt which is used to signal frame start. Earlier version of this driver used V4L2_EVENT_OMAP3ISP_HS_VS for this purpose. The event is triggered exactly when the reception of the first line of the frame starts in the CCDC module. The event can be subscribed on the CCDC subdev.

(When using parallel interface one must pay account to correct configuration of the VS signal polarity. This is automatically correct when using the serial receivers.)

Each of the statistics subdevs is able to produce events. An event is generated whenever a statistics buffer can be dequeued by a user space application using the VIDIOC_OMAP3ISP_STAT_REQ IOCTL. The events available are:

- V4L2_EVENT_OMAP3ISP_AEWB
- V4L2_EVENT_OMAP3ISP_AF
- V4L2_EVENT_OMAP3ISP_HIST

The type of the event data is struct omap3isp_stat_event_status for these ioctls. If there is an error calculating the statistics, there will be an event as usual, but no related statistics buffer. In this case omap3isp_stat_event_status.buf_err is set to non-zero.

4.17.5 Private IOCTLs

The OMAP 3 ISP driver supports standard V4L2 IOCTLs and controls where possible and practical. Much of the functions provided by the ISP, however, does not fall under the standard IOCTLs — gamma tables and configuration of statistics collection are examples of such.

In general, there is a private ioctl for configuring each of the blocks containing hardware-dependent functions.

The following private IOCTLs are supported:

- VIDIOC_OMAP3ISP_CCDC_CFG
- VIDIOC_OMAP3ISP_PRV_CFG
- VIDIOC_OMAP3ISP_AEWB_CFG

[2] http://git.ideasonboard.org/?p=media-ctl.git;a=summary

- VIDIOC_OMAP3ISP_HIST_CFG
- VIDIOC_OMAP3ISP_AF_CFG
- VIDIOC_OMAP3ISP_STAT_REQ
- VIDIOC_OMAP3ISP_STAT_EN

The parameter structures used by these ioctls are described in include/linux/omap3isp.h. The detailed functions of the ISP itself related to a given ISP block is described in the Technical Reference Manuals (TRMs) — see the end of the document for those.

While it is possible to use the ISP driver without any use of these private IOCTLs it is not possible to obtain optimal image quality this way. The AEWB, AF and histogram modules cannot be used without configuring them using the appropriate private IOCTLs.

4.17.6 CCDC and preview block IOCTLs

The VIDIOC_OMAP3ISP_CCDC_CFG and VIDIOC_OMAP3ISP_PRV_CFG IOCTLs are used to configure, enable and disable functions in the CCDC and preview blocks, respectively. Both IOCTLs control several functions in the blocks they control. VIDIOC_OMAP3ISP_CCDC_CFG IOCTL accepts a pointer to struct omap3isp_ccdc_update_config as its argument. Similarly VIDIOC_OMAP3ISP_PRV_CFG accepts a pointer to struct omap3isp_prev_update_config. The definition of both structures is available in [1].

The update field in the structures tells whether to update the configuration for the specific function and the flag tells whether to enable or disable the function.

The update and flag bit masks accept the following values. Each separate functions in the CCDC and preview blocks is associated with a flag (either disable or enable; part of the flag field in the structure) and a pointer to configuration data for the function.

Valid values for the update and flag fields are listed here for VIDIOC_OMAP3ISP_CCDC_CFG. Values may be or'ed to configure more than one function in the same IOCTL call.

- OMAP3ISP_CCDC_ALAW
- OMAP3ISP_CCDC_LPF
- OMAP3ISP_CCDC_BLCLAMP
- OMAP3ISP_CCDC_BCOMP
- OMAP3ISP_CCDC_FPC
- OMAP3ISP_CCDC_CULL
- OMAP3ISP_CCDC_CONFIG_LSC
- OMAP3ISP_CCDC_TBL_LSC

The corresponding values for the VIDIOC_OMAP3ISP_PRV_CFG are here:

- OMAP3ISP_PREV_LUMAENH
- OMAP3ISP_PREV_INVALAW
- OMAP3ISP_PREV_HRZ_MED
- OMAP3ISP_PREV_CFA
- OMAP3ISP_PREV_CHROMA_SUPP
- OMAP3ISP_PREV_WB
- OMAP3ISP_PREV_BLKADJ
- OMAP3ISP_PREV_RGB2RGB
- OMAP3ISP_PREV_COLOR_CONV

[1] include/linux/omap3isp.h

- OMAP3ISP_PREV_YC_LIMIT
- OMAP3ISP_PREV_DEFECT_COR
- OMAP3ISP_PREV_GAMMABYPASS
- OMAP3ISP_PREV_DRK_FRM_CAPTURE
- OMAP3ISP_PREV_DRK_FRM_SUBTRACT
- OMAP3ISP_PREV_LENS_SHADING
- OMAP3ISP_PREV_NF
- OMAP3ISP_PREV_GAMMA

The associated configuration pointer for the function may not be NULL when enabling the function. When disabling a function the configuration pointer is ignored.

4.17.7 Statistic blocks IOCTLs

The statistics subdevs do offer more dynamic configuration options than the other subdevs. They can be enabled, disable and reconfigured when the pipeline is in streaming state.

The statistics blocks always get the input image data from the CCDC (as the histogram memory read isn't implemented). The statistics are dequeueable by the user from the statistics subdev nodes using private IOCTLs.

The private IOCTLs offered by the AEWB, AF and histogram subdevs are heavily reflected by the register level interface offered by the ISP hardware. There are aspects that are purely related to the driver implementation and these are discussed next.

4.17.8 VIDIOC_OMAP3ISP_STAT_EN

This private IOCTL enables/disables a statistic module. If this request is done before streaming, it will take effect as soon as the pipeline starts to stream. If the pipeline is already streaming, it will take effect as soon as the CCDC becomes idle.

4.17.9 VIDIOC_OMAP3ISP_AEWB_CFG, VIDIOC_OMAP3ISP_HIST_CFG and VIDIOC_OMAP3ISP_AF_CFG

Those IOCTLs are used to configure the modules. They require user applications to have an in-depth knowledge of the hardware. Most of the fields explanation can be found on OMAP's TRMs. The two following fields common to all the above configure private IOCTLs require explanation for better understanding as they are not part of the TRM.

omap3isp_[h3a_af/h3a_aewb/hist]_config.buf_size:

The modules handle their buffers internally. The necessary buffer size for the module's data output depends on the requested configuration. Although the driver supports reconfiguration while streaming, it does not support a reconfiguration which requires bigger buffer size than what is already internally allocated if the module is enabled. It will return -EBUSY on this case. In order to avoid such condition, either disable/reconfigure/enable the module or request the necessary buffer size during the first configuration while the module is disabled.

The internal buffer size allocation considers the requested configuration's minimum buffer size and the value set on buf_size field. If buf_size field is out of [minimum, maximum] buffer size range, it's clamped to fit in there. The driver then selects the biggest value. The corrected buf_size value is written back to user application.

omap3isp_[h3a_af/h3a_aewb/hist]_config.config_counter:

As the configuration doesn't take effect synchronously to the request, the driver must provide a way to track this information to provide more accurate data. After a configuration is requested, the config_counter returned to user space application will be an unique value associated to that request. When user application receives an event for buffer availability or when a new buffer is requested, this config_counter is used to match a buffer data and a configuration.

4.17.10 VIDIOC_OMAP3ISP_STAT_REQ

Send to user space the oldest data available in the internal buffer queue and discards such buffer afterwards. The field omap3isp_stat_data.frame_number matches with the video buffer's field_count.

4.17.11 Technical reference manuals (TRMs) and other documentation

OMAP 3430 TRM: <URL:http://focus.ti.com/pdfs/wtbu/OMAP34xx_ES3.1.x_PUBLIC_TRM_vZM.zip> Referenced 2011-03-05.

OMAP 35xx TRM: <URL:http://www.ti.com/litv/pdf/spruf98o> Referenced 2011-03-05.

OMAP 3630 TRM: <URL:http://focus.ti.com/pdfs/wtbu/OMAP36xx_ES1.x_PUBLIC_TRM_vQ.zip> Referenced 2011-03-05.

DM 3730 TRM: <URL:http://www.ti.com/litv/pdf/sprugn4h> Referenced 2011-03-06.

4.17.12 References

4.18 OMAP4 ISS Driver

Author: Sergio Aguirre <sergio.a.aguirre@gmail.com>

4.18.1 Introduction

The OMAP44XX family of chips contains the Imaging SubSystem (a.k.a. ISS), Which contains several components that can be categorized in 3 big groups:

- Interfaces (2 Interfaces: CSI2-A & CSI2-B/CCP2)
- ISP (Image Signal Processor)
- SIMCOP (Still Image Coprocessor)

For more information, please look in [1] for latest version of: "OMAP4430 Multimedia Device Silicon Revision 2.x"

As of Revision AB, the ISS is described in detail in section 8.

This driver is supporting **only** the CSI2-A/B interfaces for now.

It makes use of the Media Controller framework [2], and inherited most of the code from OMAP3 ISP driver (found under drivers/media/platform/omap3isp/*), except that it doesn't need an IOMMU now for ISS buffers memory mapping.

Supports usage of MMAP buffers only (for now).

[1] http://focus.ti.com/general/docs/wtbu/wtbudocumentcenter.tsp?navigationId=12037&templateId=6123#62
[2] http://lwn.net/Articles/420485/

4.18.2 Tested platforms

- OMAP4430SDP, w/ ES2.1 GP & SEVM4430-CAM-V1-0 (Contains IMX060 & OV5640, in which only the last one is supported, outputting YUV422 frames).
- TI Blaze MDP, w/ OMAP4430 ES2.2 EMU (Contains 1 IMX060 & 2 OV5650 sensors, in which only the OV5650 are supported, outputting RAW10 frames).
- PandaBoard, Rev. A2, w/ OMAP4430 ES2.1 GP & OV adapter board, tested with following sensors: * OV5640 * OV5650
- Tested on mainline kernel:

  http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=summary

  Tag: v3.3 (commit c16fa4f2ad19908a47c63d8fa436a1178438c7e7)

4.18.3 File list

drivers/staging/media/omap4iss/ include/linux/platform_data/media/omap4iss.h

4.18.4 References

4.19 The pvrusb2 driver

Author: Mike Isely <isely@pobox.com>

4.19.1 Background

This driver is intended for the "Hauppauge WinTV PVR USB 2.0", which is a USB 2.0 hosted TV Tuner. This driver is a work in progress. Its history started with the reverse-engineering effort by Björn Danielsson <pvrusb2@dax.nu> whose web page can be found here: http://pvrusb2.dax.nu/

From there Aurelien Alleaume <slts@free.fr> began an effort to create a video4linux compatible driver. I began with Aurelien's last known snapshot and evolved the driver to the state it is in here.

More information on this driver can be found at: http://www.isely.net/pvrusb2.html

This driver has a strong separation of layers. They are very roughly:

1. Low level wire-protocol implementation with the device.
2. I2C adaptor implementation and corresponding I2C client drivers implemented elsewhere in V4L.
3. High level hardware driver implementation which coordinates all activities that ensure correct operation of the device.
4. A "context" layer which manages instancing of driver, setup, tear-down, arbitration, and interaction with high level interfaces appropriately as devices are hotplugged in the system.
5. High level interfaces which glue the driver to various published Linux APIs (V4L, sysfs, maybe DVB in the future).

The most important shearing layer is between the top 2 layers. A lot of work went into the driver to ensure that any kind of conceivable API can be laid on top of the core driver. (Yes, the driver internally leverages V4L to do its work but that really has nothing to do with the API published by the driver to the outside world.) The architecture allows for different APIs to simultaneously access the driver. I have a strong sense of fairness about APIs and also feel that it is a good design principle to keep implementation and interface isolated from each other. Thus while right now the V4L high level interface is the most complete, the sysfs high level interface will work equally well for similar functions, and there's no reason I see right now why it shouldn't be possible to produce a DVB high level interface that can sit right alongside V4L.

4.19.2 Building

To build these modules essentially amounts to just running "Make", but you need the kernel source tree nearby and you will likely also want to set a few controlling environment variables first in order to link things up with that source tree. Please see the Makefile here for comments that explain how to do that.

4.19.3 Source file list / functional overview

(Note: The term "module" used below generally refers to loosely defined functional units within the pvrusb2 driver and bears no relation to the Linux kernel's concept of a loadable module.)

pvrusb2-audio.[ch] - This is glue logic that resides between this driver and the msp3400.ko I2C client driver (which is found elsewhere in V4L).

pvrusb2-context.[ch] - This module implements the context for an instance of the driver. Everything else eventually ties back to or is otherwise instanced within the data structures implemented here. Hotplugging is ultimately coordinated here. All high level interfaces tie into the driver through this module. This module helps arbitrate each interface's access to the actual driver core, and is designed to allow concurrent access through multiple instances of multiple interfaces (thus you can for example change the tuner's frequency through sysfs while simultaneously streaming video through V4L out to an instance of mplayer).

pvrusb2-debug.h - This header defines a printk() wrapper and a mask of debugging bit definitions for the various kinds of debug messages that can be enabled within the driver.

pvrusb2-debugifc.[ch] - This module implements a crude command line oriented debug interface into the driver. Aside from being part of the process for implementing manual firmware extraction (see the pvrusb2 web site mentioned earlier), probably I'm the only one who has ever used this. It is mainly a debugging aid.

pvrusb2-eeprom.[ch] - This is glue logic that resides between this driver the tveeprom.ko module, which is itself implemented elsewhere in V4L.

pvrusb2-encoder.[ch] - This module implements all protocol needed to interact with the Conexant mpeg2 encoder chip within the pvrusb2 device. It is a crude echo of corresponding logic in ivtv, however the design goals (strict isolation) and physical layer (proxy through USB instead of PCI) are enough different that this implementation had to be completely different.

pvrusb2-hdw-internal.h - This header defines the core data structure in the driver used to track ALL internal state related to control of the hardware. Nobody outside of the core hardware-handling modules should have any business using this header. All external access to the driver should be through one of the high level interfaces (e.g. V4L, sysfs, etc), and in fact even those high level interfaces are restricted to the API defined in pvrusb2-hdw.h and NOT this header.

pvrusb2-hdw.h - This header defines the full internal API for controlling the hardware. High level interfaces (e.g. V4L, sysfs) will work through here.

pvrusb2-hdw.c - This module implements all the various bits of logic that handle overall control of a specific pvrusb2 device. (Policy, instantiation, and arbitration of pvrusb2 devices fall within the jurisdiction of pvrusb-context not here).

pvrusb2-i2c-chips-*.c - These modules implement the glue logic to tie together and configure various I2C modules as they attach to the I2C bus. There are two versions of this file. The "v4l2" version is intended to be used in-tree alongside V4L, where we implement just the logic that makes sense for a pure V4L environment. The "all" version is intended for use outside of V4L, where we might encounter other possibly "challenging" modules from ivtv or older kernel snapshots (or even the support modules in the standalone snapshot).

pvrusb2-i2c-cmd-v4l1.[ch] - This module implements generic V4L1 compatible commands to the I2C modules. It is here where state changes inside the pvrusb2 driver are translated into V4L1 commands that are in turn send to the various I2C modules.

pvrusb2-i2c-cmd-v4l2.[ch] - This module implements generic V4L2 compatible commands to the I2C modules. It is here where state changes inside the pvrusb2 driver are translated into V4L2 commands that are in turn send to the various I2C modules.

pvrusb2-i2c-core.[ch] - This module provides an implementation of a kernel-friendly I2C adaptor driver, through which other external I2C client drivers (e.g. msp3400, tuner, lirc) may connect and operate corresponding chips within the pvrusb2 device. It is through here that other V4L modules can reach into this driver to operate specific pieces (and those modules are in turn driven by glue logic which is coordinated by pvrusb2-hdw, doled out by pvrusb2-context, and then ultimately made available to users through one of the high level interfaces).

pvrusb2-io.[ch] - This module implements a very low level ring of transfer buffers, required in order to stream data from the device. This module is *very* low level. It only operates the buffers and makes no attempt to define any policy or mechanism for how such buffers might be used.

pvrusb2-ioread.[ch] - This module layers on top of pvrusb2-io.[ch] to provide a streaming API usable by a read() system call style of I/O. Right now this is the only layer on top of pvrusb2-io.[ch], however the underlying architecture here was intended to allow for other styles of I/O to be implemented with additional modules, like mmap()'ed buffers or something even more exotic.

pvrusb2-main.c - This is the top level of the driver. Module level and USB core entry points are here. This is our "main".

pvrusb2-sysfs.[ch] - This is the high level interface which ties the pvrusb2 driver into sysfs. Through this interface you can do everything with the driver except actually stream data.

pvrusb2-tuner.[ch] - This is glue logic that resides between this driver and the tuner.ko I2C client driver (which is found elsewhere in V4L).

pvrusb2-util.h - This header defines some common macros used throughout the driver. These macros are not really specific to the driver, but they had to go somewhere.

pvrusb2-v4l2.[ch] - This is the high level interface which ties the pvrusb2 driver into video4linux. It is through here that V4L applications can open and operate the driver in the usual V4L ways. Note that **ALL** V4L functionality is published only through here and nowhere else.

pvrusb2-video-*.[ch] - This is glue logic that resides between this driver and the saa711x.ko I2C client driver (which is found elsewhere in V4L). Note that saa711x.ko used to be known as saa7115.ko in ivtv. There are two versions of this; one is selected depending on the particular saa711[5x].ko that is found.

pvrusb2.h - This header contains compile time tunable parameters (and at the moment the driver has very little that needs to be tuned).

4.20 PXA-Camera Host Driver

Author: Robert Jarzmik <robert.jarzmik@free.fr>

4.20.1 Constraints

1. Image size for YUV422P format All YUV422P images are enforced to have width x height % 16 = 0. This is due to DMA constraints, which transfers only planes of 8 byte multiples.

4.20.2 Global video workflow

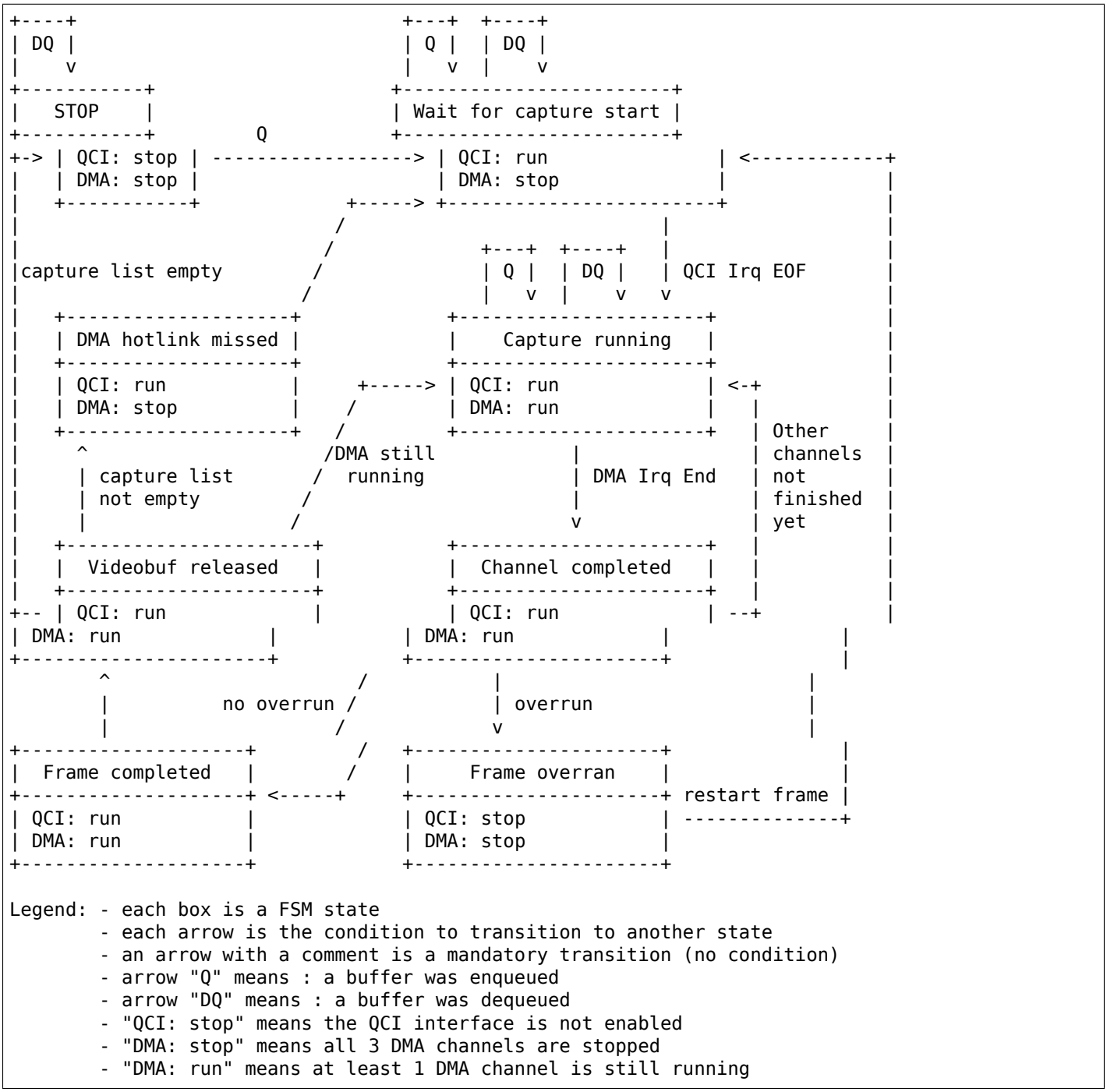
1. QCI stopped Initialy, the QCI interface is stopped. When a buffer is queued (pxa_videobuf_ops->buf_queue), the QCI starts.

2. QCI started More buffers can be queued while the QCI is started without halting the capture. The new buffers are “appended” at the tail of the DMA chain, and smoothly captured one frame after the other.

   Once a buffer is filled in the QCI interface, it is marked as “DONE” and removed from the active buffers list. It can be then requeud or dequeued by userland application.

   Once the last buffer is filled in, the QCI interface stops.

3. Capture global finite state machine schema



4.20.3 DMA usage

1. **DMA flow**

- first buffer queued for capture Once a first buffer is queued for capture, the QCI is started, but data transfer is not started. On “End Of Frame” interrupt, the irq handler starts the DMA chain.
- capture of one videobuffer The DMA chain starts transferring data into videobuffer RAM pages. When all pages are transferred, the DMA irq is raised on “ENDINTR” status
- finishing one videobuffer The DMA irq handler marks the videobuffer as “done”, and removes it from the active running queue Meanwhile, the next videobuffer (if there is one), is transferred by DMA
- finishing the last videobuffer On the DMA irq of the last videobuffer, the QCI is stopped.

2. DMA prepared buffer will have this structure

```
+------------+-----+---------------+-----------------+
| desc-sg[0] | ... | desc-sg[last] | finisher/linker |
+------------+-----+---------------+-----------------+
```

This structure is pointed by dma->sg_cpu. The descriptors are used as follows:

- desc-sg[i]: i-th descriptor, transferring the i-th sg element to the video buffer scatter gather
- finisher: has ddadr=DADDR_STOP, dcmd=ENDIRQEN
- linker: has ddadr= desc-sg[0] of next video buffer, dcmd=0

For the next schema, let’s assume d0=desc-sg[0] .. dN=desc-sg[N], “f” stands for finisher and “l” for linker. A typical running chain is :

```
    Videobuffer 1          Videobuffer 2
+---------+----+---+  +----+----+----+---+
| d0 | .. | dN | l |  | d0 | .. | dN | f |
+---------+----+-|-+  ^----+----+----+---+
                 |    |
                 +----+
```

After the chaining is finished, the chain looks like :

```
    Videobuffer 1          Videobuffer 2          Videobuffer 3
+---------+----+---+  +----+----+----+---+  +----+----+----+---+
| d0 | .. | dN | l |  | d0 | .. | dN | l |  | d0 | .. | dN | f |
+---------+----+-|-+  ^----+----+----+-|-+  ^----+----+----+---+
                 |    |                |    |
                 +----+                +----+
                                     new_link
```

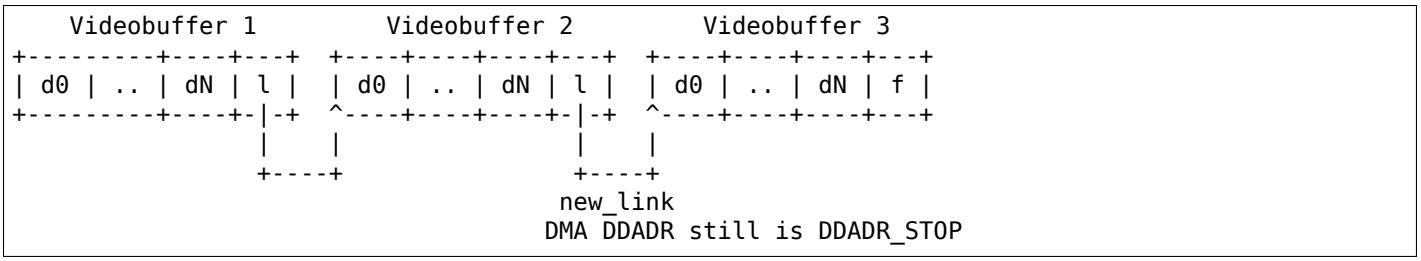
3. DMA hot chaining timeslice issue

As DMA chaining is done while DMA _is_ running, the linking may be done while the DMA jumps from one Videobuffer to another. On the schema, that would be a problem if the following sequence is encountered :

- DMA chain is Videobuffer1 + Videobuffer2
- pxa_videobuf_queue() is called to queue Videobuffer3
- DMA controller finishes Videobuffer2, and DMA stops

```
 =>
    Videobuffer 1          Videobuffer 2
+---------+----+---+  +----+----+----+---+
| d0 | .. | dN | l |  | d0 | .. | dN | f |
+---------+----+-|-+  ^----+----+----+-^-+
                 |    |                |
                 +----+                +-- DMA DDADR loads DDADR_STOP
```

- pxa_dma_add_tail_buf() is called, the Videobuffer2 “finisher” is replaced by a “linker” to Videobuffer3 (creation of new_link)
- pxa_videobuf_queue() finishes
- the DMA irq handler is called, which terminates Videobuffer2
- Videobuffer3 capture is not scheduled on DMA chain (as it stopped !!!)



- pxa_camera_check_link_miss() is called This checks if the DMA is finished and a buffer is still on the pcdev->capture list. If that’s the case, the capture will be restarted, and Videobuffer3 is scheduled on DMA chain.
- the DMA irq handler finishes

Note:

If DMA stops just after pxa_camera_check_link_miss() reads DDADR() value, we have the guarantee that the DMA irq handler will be called back when the DMA will finish the buffer, and pxa_camera_check_link_miss() will be called again, to reschedule Videobuffer3.

4.21 The Radiotrack radio driver

Author: Stephen M. Benoit <benoits@servicepro.com>

Date: Dec 14, 1996

4.21.1 ACKNOWLEDGMENTS

This document was made based on ‘C’ code for Linux from Gideon le Grange (legrang@active.co.za or legrang@cs.sun.ac.za) in 1994, and elaborations from Frans Brinkman (brinkman@esd.nl) in 1996. The results reported here are from experiments that the author performed on his own setup, so your mileage may vary... I make no guarantees, claims or warranties to the suitability or validity of this information. No other documentation on the AIMS Lab (http://www.aimslab.com/) RadioTrack card was made available to the author. This document is offered in the hopes that it might help users who want to use the RadioTrack card in an environment other than MS Windows.

4.21.2 WHY THIS DOCUMENT?

I have a RadioTrack card from back when I ran an MS-Windows platform. After converting to Linux, I found Gideon le Grange’s command-line software for running the card, and found that it was good! Frans Brinkman made a comfortable X-windows interface, and added a scanning feature. For hack value, I wanted to see if the tuner could be tuned beyond the usual FM radio broadcast band, so I could pick up the audio carriers from North American broadcast TV channels, situated just below and above the 87.0-109.0 MHz range. I did not get much success, but I learned about programming ioports under Linux and gained some insights about the hardware design used for the card.

So, without further delay, here are the details.

4.21.3 PHYSICAL DESCRIPTION

The RadioTrack card is an ISA 8-bit FM radio card. The radio frequency (RF) input is simply an antenna lead, and the output is a power audio signal available through a miniature phone plug. Its RF frequencies of operation are more or less limited from 87.0 to 109.0 MHz (the commercial FM broadcast band). Although the registers can be programmed to request frequencies beyond these limits, experiments did not give promising results. The variable frequency oscillator (VFO) that demodulates the intermediate frequency (IF) signal probably has a small range of useful frequencies, and wraps around or gets clipped beyond the limits mentioned above.

4.21.4 CONTROLLING THE CARD WITH IOPORT

The RadioTrack (base) ioport is configurable for 0x30c or 0x20c. Only one ioport seems to be involved. The ioport decoding circuitry must be pretty simple, as individual ioport bits are directly matched to specific functions (or blocks) of the radio card. This way, many functions can be changed in parallel with one write to the ioport. The only feedback available through the ioports appears to be the “Stereo Detect” bit.

The bits of the ioport are arranged as follows:

```
MSb                                                             LSb
+------+------+------+--------+--------+-------+---------+--------+
| VolA | VolB | ???? | Stereo | Radio  | TuneA | TuneB   | Tune   |
|  (+) |  (-) |      | Detect | Audio  | (bit) | (latch) | Update |
|      |      |      | Enable | Enable |       |         | Enable |
+------+------+------+--------+--------+-------+---------+--------+
```

VolA	VolB	Description
0	0	audio mute
0	1	volume + (some delay required)
1	0	volume - (some delay required)
1	1	stay at present volume

Stereo Detect Enable	Description
0	No Detect
1	Detect

Results available by reading ioport >60 msec after last port write.

0xff ==> no stereo detected, 0xfd ==> stereo detected.

Radio to Audio (path) Enable	Description
0	Disable path (silence)
1	Enable path (audio produced)

TuneA	TuneB	Description
0	0	“zero” bit phase 1
0	1	“zero” bit phase 2
1	0	“one” bit phase 1
1	1	“one” bit phase 2

24-bit code, where bits = (freq*40) + 10486188. The Most Significant 11 bits must be 1010 xxxx 0x0 to be valid. The bits are shifted in LSb first.

Tune Update Enable	Description
0	Tuner held constant
1	Tuner updating in progress

4.21.5 PROGRAMMING EXAMPLES

```
Default:        BASE <-- 0xc8  (current volume, no stereo detect,
                                radio enable, tuner adjust disable)

Card Off:       BASE <-- 0x00  (audio mute, no stereo detect,
                                radio disable, tuner adjust disable)

Card On:        BASE <-- 0x00  (see "Card Off", clears any unfinished business)
                BASE <-- 0xc8  (see "Default")

Volume Down:    BASE <-- 0x48  (volume down, no stereo detect,
                                radio enable, tuner adjust disable)
                wait 10 msec
                BASE <-- 0xc8  (see "Default")

Volume Up:      BASE <-- 0x88  (volume up, no stereo detect,
                                radio enable, tuner adjust disable)
                wait 10 msec
                BASE <-- 0xc8  (see "Default")

Check Stereo:   BASE <-- 0xd8  (current volume, stereo detect,
                                radio enable, tuner adjust disable)
                wait 100 msec
                x <-- BASE     (read ioport)
                BASE <-- 0xc8  (see "Default")

                x=0xff ==> "not stereo", x=0xfd ==> "stereo detected"

Set Frequency:  code = (freq*40) + 10486188
                foreach of the 24 bits in code,
                (from Least to Most Significant):
                to write a "zero" bit,
                BASE <-- 0x01  (audio mute, no stereo detect, radio
                                disable, "zero" bit phase 1, tuner adjust)
                BASE <-- 0x03  (audio mute, no stereo detect, radio
                                disable, "zero" bit phase 2, tuner adjust)
                to write a "one" bit,
                BASE <-- 0x05  (audio mute, no stereo detect, radio
                                disable, "one" bit phase 1, tuner adjust)
                BASE <-- 0x07  (audio mute, no stereo detect, radio
                                disable, "one" bit phase 2, tuner adjust)
```

4.22 Renesas R-Car Fine Display Processor (FDP1) Driver

The R-Car FDP1 driver implements driver-specific controls as follows.

V4L2_CID_DEINTERLACING_MODE (menu) The video deinterlacing mode (such as Bob, Weave, ...). The R-Car FDP1 driver implements the following modes.

"Progressive" (0)	The input image video stream is progressive (not interlaced). No deinterlacing is performed. Apart from (optional) format and encoding conversion output frames are identical to the input frames.
"Adaptive 2D/3D" (1)	Motion adaptive version of 2D and 3D deinterlacing. Use 3D deinterlacing in the presence of fast motion and 2D deinterlacing with diagonal interpolation otherwise.
"Fixed 2D" (2)	The current field is scaled vertically by averaging adjacent lines to recover missing lines. This method is also known as blending or Line Averaging (LAV).
"Fixed 3D" (3)	The previous and next fields are averaged to recover lines missing from the current field. This method is also known as Field Averaging (FAV).
"Previous field" (4)	The current field is weaved with the previous field, i.e. the previous field is used to fill missing lines from the current field. This method is also known as weave deinterlacing.
"Next field" (5)	The current field is weaved with the next field, i.e. the next field is used to fill missing lines from the current field. This method is also known as weave deinterlacing.

4.23 The saa7134 driver

Author Gerd Hoffmann

This is a v4l2/oss device driver for saa7130/33/34/35 based capture / TV boards. See http://www.semiconductors.philips.com/pip/saa7134hl for a description.

4.23.1 Status

Almost everything is working. video, sound, tuner, radio, mpeg ts, ...

As with bttv, card-specific tweaks are needed. Check CARDLIST for a list of known TV cards and saa7134-cards.c for the drivers card configuration info.

4.23.2 Build

Pick up videodev + v4l2 patches from http://bytesex.org/patches/. Configure, build, install + boot the new kernel. You'll need at least these config options:

```
CONFIG_I2C=m
CONFIG_VIDEO_DEV=m
```

Type "make" to build the driver now. "make install" installs the driver. "modprobe saa7134" should load it. Depending on the card you might have to pass card=<nr> as insmod option, check CARDLIST for valid choices.

4.23.3 Changes / Fixes

Please mail me unified diffs ("diff -u") with your changes, and don't forget to tell me what it changes / which problem it fixes / whatever it is good for ...

4.23.4 Known Problems

- The tuner for the flyvideos isn't detected automatically and the default might not work for you depending on which version you have. There is a tuner= insmod option to override the driver's default.

4.23.5 Card Variations:

Cards can use either of these two crystals (xtal):

- 32.11 MHz -> .audio_clock=0x187de7
- 24.576MHz -> .audio_clock=0x200000 (xtal * .audio_clock = 51539600)

Some details about 30/34/35:

- saa7130 - low-price chip, doesn't have mute, that is why all those cards should have .mute field defined in their tuner structure.
- saa7134 - usual chip
- saa7133/35 - saa7135 is probably a marketing decision, since all those chips identifies itself as 33 on pci.

4.23.6 LifeView GPIOs

This section was authored by: Peter Missel <peter.missel@onlinehome.de>

- LifeView FlyTV Platinum FM (LR214WF)
  - GP27 MDT2005 PB4 pin 10
  - GP26 MDT2005 PB3 pin 9
  - GP25 MDT2005 PB2 pin 8
  - GP23 MDT2005 PB1 pin 7
  - GP22 MDT2005 PB0 pin 6
  - GP21 MDT2005 PB5 pin 11
  - GP20 MDT2005 PB6 pin 12
  - GP19 MDT2005 PB7 pin 13
  - nc MDT2005 PA3 pin 2
  - Remote MDT2005 PA2 pin 1
  - GP18 MDT2005 PA1 pin 18
  - nc MDT2005 PA0 pin 17 strap low
  - GP17 Strap "GP7"=High
  - GP16 Strap "GP6"=High
    - 0=Radio 1=TV
    - Drives SA630D ENCH1 and HEF4052 A1 pinsto do FM radio through SIF input
  - GP15 nc
  - GP14 nc
  - GP13 nc
  - GP12 Strap "GP5" = High
  - GP11 Strap "GP4" = High
  - GP10 Strap "GP3" = High
  - GP09 Strap "GP2" = Low
  - GP08 Strap "GP1" = Low
  - GP07.00 nc

4.23.7 Credits

andrew.stevens@philips.com + werner.leeb@philips.com for providing saa7134 hardware specs and sample board.

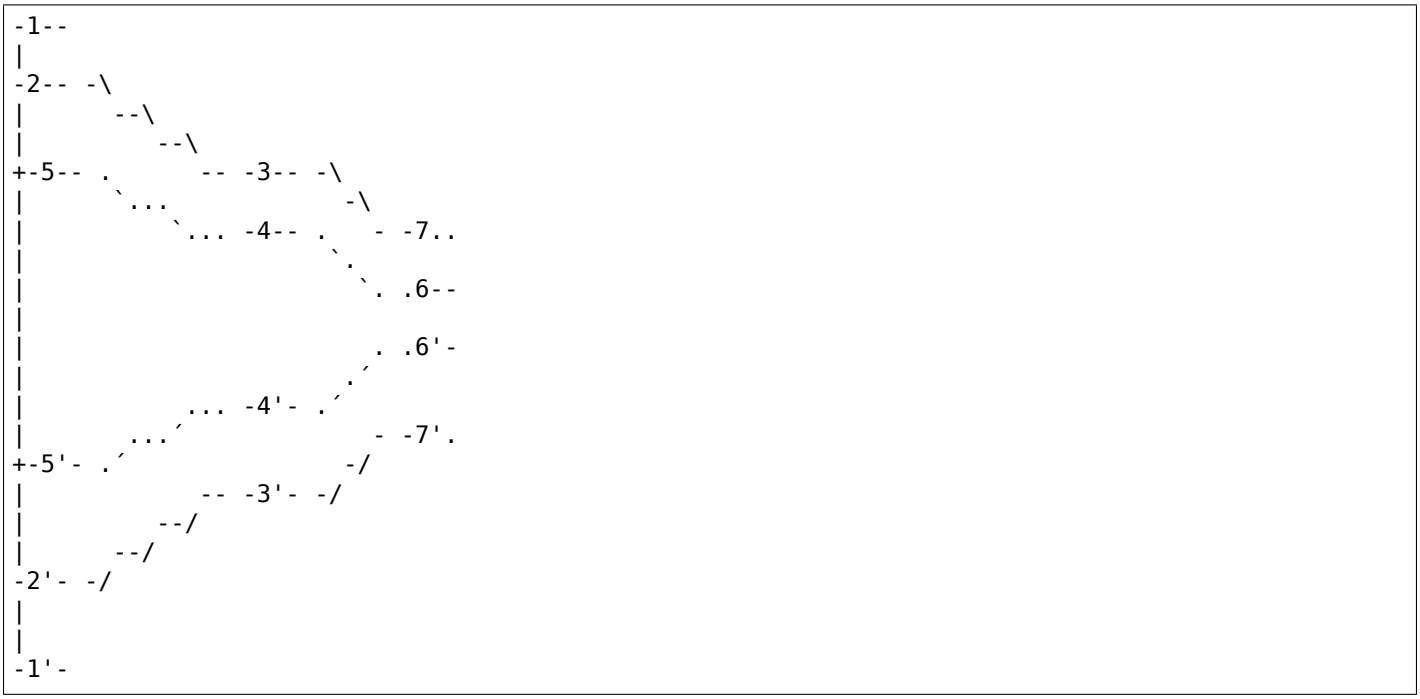
4.24 Cropping and Scaling algorithm, used in the sh_mobile_ceu_camera driver

Author: Guennadi Liakhovetski <g.liakhovetski@gmx.de>

4.24.1 Terminology

sensor scales: horizontal and vertical scales, configured by the sensor driver host scales: -“- host driver combined scales: sensor_scale * host_scale

4.24.2 Generic scaling / cropping scheme



In the above chart minuses and slashes represent “real” data amounts, points and accents represent “useful” data, basically, CEU scaled and cropped output, mapped back onto the client’s source plane.

Such a configuration can be produced by user requests:

S_CROP(left / top = (5) - (1), width / height = (5’) - (5)) S_FMT(width / height = (6’) - (6))

Here:

(1) to (1’) - whole max width or height (1) to (2) - sensor cropped left or top (2) to (2’) - sensor cropped width or height (3) to (3’) - sensor scale (3) to (4) - CEU cropped left or top (4) to (4’) - CEU cropped width or height (5) to (5’) - reverse sensor scale applied to CEU cropped width or height (2) to (5) - reverse sensor scale applied to CEU cropped left or top (6) to (6’) - CEU scale - user window

4.24.3 S_FMT

Do not touch input rectangle - it is already optimal.

1. Calculate current sensor scales:

   scale_s = ((2') - (2)) / ((3') - (3))

2. Calculate "effective" input crop (sensor subwindow) - CEU crop scaled back at current sensor scales onto input window - this is user S_CROP:

   width_u = (5') - (5) = ((4') - (4)) * scale_s

3. Calculate new combined scales from "effective" input window to requested user window:

   scale_comb = width_u / ((6') - (6))

4. Calculate sensor output window by applying combined scales to real input window:

   width_s_out = ((7') - (7)) = ((2') - (2)) / scale_comb

5. Apply iterative sensor S_FMT for sensor output window.

   subdev->video_ops->s_fmt(.width = width_s_out)

6. Retrieve sensor output window (g_fmt)

7. Calculate new sensor scales:

   scale_s_new = ((3')_new - (3)_new) / ((2') - (2))

8. Calculate new CEU crop - apply sensor scales to previously calculated "effective" crop:

   width_ceu = (4')_new - (4)_new = width_u / scale_s_new left_ceu = (4)_new - (3)_new = ((5) - (2)) / scale_s_new

9. Use CEU cropping to crop to the new window:

   ceu_crop(.width = width_ceu, .left = left_ceu)

10. Use CEU scaling to scale to the requested user window:

    scale_ceu = width_ceu / width

4.24.4 S_CROP

The API at http://v4l2spec.bytesex.org/spec/x1904.htm says:

"...specification does not define an origin or units. However by convention drivers should horizontally count unscaled samples relative to 0H."

We choose to follow the advise and interpret cropping units as client input pixels.

Cropping is performed in the following 6 steps:

1. Request exactly user rectangle from the sensor.
2. If smaller - iterate until a larger one is obtained. Result: sensor cropped to 2 : 2', target crop 5 : 5', current output format 6' - 6.
3. In the previous step the sensor has tried to preserve its output frame as good as possible, but it could have changed. Retrieve it again.
4. Sensor scaled to 3 : 3'. Sensor's scale is (2' - 2) / (3' - 3). Calculate intermediate window: 4' - 4 = (5' - 5) * (3' - 3) / (2' - 2)
5. Calculate and apply host scale = (6' - 6) / (4' - 4)
6. Calculate and apply host crop: 6 - 7 = (5 - 2) * (6' - 6) / (5' - 5)

4.25 The Silicon Labs Si470x FM Radio Receivers driver

4.25.1 Information from Silicon Labs

Silicon Laboratories is the manufacturer of the radio ICs, that nowadays are the most often used radio receivers in cell phones. Usually they are connected with I2C. But SiLabs also provides a reference design, which integrates this IC, together with a small microcontroller C8051F321, to form a USB radio. Part of this reference design is also a radio application in binary and source code. The software also contains an automatic firmware upgrade to the most current version. Information on these can be downloaded here: http://www.silabs.com/usbradio

4.25.2 Supported ICs

The following ICs have a very similar register set, so that they are or will be supported somewhen by the driver:

- Si4700: FM radio receiver
- Si4701: FM radio receiver, RDS Support
- Si4702: FM radio receiver
- Si4703: FM radio receiver, RDS Support
- Si4704: FM radio receiver, no external antenna required
- Si4705: FM radio receiver, no external antenna required, RDS support, Dig I/O
- **Si4706: Enhanced FM RDS/TMC radio receiver, no external antenna required, RDS** Support
- Si4707: Dedicated weather band radio receiver with SAME decoder, RDS Support
- Si4708: Smallest FM receivers
- Si4709: Smallest FM receivers, RDS Support

More information on these can be downloaded here: http://www.silabs.com/products/mcu/Pages/USBFMRadioRD.aspx

4.25.3 Supported USB devices

Currently the following USB radios (vendor:product) with the Silicon Labs si470x chips are known to work:

- 10c4:818a: Silicon Labs USB FM Radio Reference Design
- 06e1:a155: ADS/Tech FM Radio Receiver (formerly Instant FM Music) (RDX-155-EF)
- 1b80:d700: KWorld USB FM Radio SnapMusic Mobile 700 (FM700)
- 10c5:819a: Sanei Electric, Inc. FM USB Radio (sold as DealExtreme.com PCear)

4.25.4 Software

Testing is usually done with most application under Debian/testing:

- fmtools - Utility for managing FM tuner cards
- gnomeradio - FM-radio tuner for the GNOME desktop
- gradio - GTK FM radio tuner

- kradio - Comfortable Radio Application for KDE
- radio - ncurses-based radio application
- mplayer - The Ultimate Movie Player For Linux
- v4l2-ctl - Collection of command line video4linux utilities

For example, you can use:

```
v4l2-ctl -d /dev/radio0 --set-ctrl=volume=10,mute=0 --set-freq=95.21 --all
```

There is also a library libv4l, which can be used. It's going to have a function for frequency seeking, either by using hardware functionality as in radio-si470x or by implementing a function as we currently have in every of the mentioned programs. Somewhen the radio programs should make use of libv4l.

For processing RDS information, there is a project ongoing at: http://rdsd.berlios.de/

There is currently no project for making TMC sentences human readable.

4.25.5 Audio Listing

USB Audio is provided by the ALSA snd_usb_audio module. It is recommended to also select SND_USB_AUDIO, as this is required to get sound from the radio. For listing you have to redirect the sound, for example using one of the following commands. Please adjust the audio devices to your needs (/dev/dsp* and hw:x,x).

If you just want to test audio (very poor quality):

```
cat /dev/dsp1 > /dev/dsp
```

If you use sox + OSS try:

```
sox -2 --endian little -r 96000 -t oss /dev/dsp1 -t oss /dev/dsp
```

or using sox + alsa:

```
sox --endian little -c 2 -S -r 96000 -t alsa hw:1 -t alsa -r 96000 hw:0
```

If you use arts try:

```
arecord -D hw:1,0 -r96000 -c2 -f S16_LE | artsdsp aplay -B -
```

If you use mplayer try:

```
mplayer -radio adevice=hw=1.0:arate=96000 \
        -rawaudio rate=96000 \
        radio://<frequency>/capture
```

4.25.6 Module Parameters

After loading the module, you still have access to some of them in the sysfs mount under /sys/module/radio_si470x/parameters. The contents of read-only files (0444) are not updated, even if space, band and de are changed using private video controls. The others are runtime changeable.

4.25.7 Errors

Increase tune_timeout, if you often get -EIO errors.

When timed out or band limit is reached, hw_freq_seek returns -EAGAIN.

If you get any errors from snd_usb_audio, please report them to the ALSA people.

4.25.8 Open Issues

V4L minor device allocation and parameter setting is not perfect. A solution is currently under discussion.

There is an USB interface for downloading/uploading new firmware images. Support for it can be implemented using the request_firmware interface.

There is a RDS interrupt mode. The driver is already using the same interface for polling RDS information, but is currently not using the interrupt mode.

There is a LED interface, which can be used to override the LED control programmed in the firmware. This can be made available using the LED support functions in the kernel.

4.25.9 Other useful information and links

http://www.silabs.com/usbradio

4.26 The Silicon Labs Si4713 FM Radio Transmitter Driver

Contact: Eduardo Valentin <eduardo.valentin@nokia.com>

4.26.1 Information about the Device

This chip is a Silicon Labs product. It is a I2C device, currently on 0x63 address. Basically, it has transmission and signal noise level measurement features.

The Si4713 integrates transmit functions for FM broadcast stereo transmission. The chip also allows integrated receive power scanning to identify low signal power FM channels.

The chip is programmed using commands and responses. There are also several properties which can change the behavior of this chip.

Users must comply with local regulations on radio frequency (RF) transmission.

4.26.2 Device driver description

There are two modules to handle this device. One is a I2C device driver and the other is a platform driver.

The I2C device driver exports a v4l2-subdev interface to the kernel. All properties can also be accessed by v4l2 extended controls interface, by using the v4l2-subdev calls (g_ext_ctrls, s_ext_ctrls).

The platform device driver exports a v4l2 radio device interface to user land. So, it uses the I2C device driver as a sub device in order to send the user commands to the actual device. Basically it is a wrapper to the I2C device driver.

Applications can use v4l2 radio API to specify frequency of operation, mute state, etc. But mostly of its properties will be present in the extended controls.

When the v4l2 mute property is set to 1 (true), the driver will turn the chip off.

4.26.3 Properties description

The properties can be accessed using v4l2 extended controls. Here is an output from v4l2-ctl util:

```
/ # v4l2-ctl -d /dev/radio0 --all -L
Driver Info:
        Driver name   : radio-si4713
        Card type     : Silicon Labs Si4713 Modulator
        Bus info      :
        Driver version: 0
        Capabilities  : 0x00080800
                RDS Output
                Modulator
Audio output: 0 (FM Modulator Audio Out)
Frequency: 1408000 (88.000000 MHz)
Video Standard = 0x00000000
Modulator:
        Name                 : FM Modulator
        Capabilities         : 62.5 Hz stereo rds
        Frequency range      : 76.0 MHz - 108.0 MHz
        Subchannel modulation: stereo+rds

User Controls

                         mute (bool) : default=1 value=0

FM Radio Modulator Controls

        rds_signal_deviation (int)  : min=0 max=90000 step=10 default=200 value=200 flags=slider
                rds_program_id (int)  : min=0 max=65535 step=1 default=0 value=0
        rds_program_type (int)  : min=0 max=31 step=1 default=0 value=0
                rds_ps_name (str)  : min=0 max=96 step=8 value='si4713  '
                rds_radio_text (str)  : min=0 max=384 step=32 value=''
audio_limiter_feature_enabled (bool) : default=1 value=1
audio_limiter_release_time (int)  : min=250 max=102390 step=50 default=5010 value=5010␣
 ↪flags=slider
        audio_limiter_deviation (int)  : min=0 max=90000 step=10 default=66250 value=66250␣
 ↪flags=slider
audio_compression_feature_enabl (bool) : default=1 value=1
        audio_compression_gain (int)  : min=0 max=20 step=1 default=15 value=15 flags=slider
audio_compression_threshold (int)  : min=-40 max=0 step=1 default=-40 value=-40 flags=slider
audio_compression_attack_time (int)  : min=0 max=5000 step=500 default=0 value=0 flags=slider
audio_compression_release_time (int)  : min=100000 max=1000000 step=100000 default=1000000␣
 ↪value=1000000 flags=slider
pilot_tone_feature_enabled (bool) : default=1 value=1
        pilot_tone_deviation (int)  : min=0 max=90000 step=10 default=6750 value=6750␣
 ↪flags=slider
        pilot_tone_frequency (int)  : min=0 max=19000 step=1 default=19000 value=19000␣
 ↪flags=slider
        pre_emphasis_settings (menu) : min=0 max=2 default=1 value=1
        tune_power_level (int)  : min=0 max=120 step=1 default=88 value=88 flags=slider
        tune_antenna_capacitor (int)  : min=0 max=191 step=1 default=0 value=110 flags=slider
```

Here is a summary of them:

- Pilot is an audible tone sent by the device.
- pilot_frequency - Configures the frequency of the stereo pilot tone.
- pilot_deviation - Configures pilot tone frequency deviation level.
- pilot_enabled - Enables or disables the pilot tone feature.
- The si4713 device is capable of applying audio compression to the transmitted signal.
- acomp_enabled - Enables or disables the audio dynamic range control feature.
- acomp_gain - Sets the gain for audio dynamic range control.

- acomp_threshold - Sets the threshold level for audio dynamic range control.
- acomp_attack_time - Sets the attack time for audio dynamic range control.
- acomp_release_time - Sets the release time for audio dynamic range control.
- Limiter setups audio deviation limiter feature. Once a over deviation occurs, it is possible to adjust the front-end gain of the audio input and always prevent over deviation.
- limiter_enabled - Enables or disables the limiter feature.
- limiter_deviation - Configures audio frequency deviation level.
- limiter_release_time - Sets the limiter release time.
- Tuning power
- power_level - Sets the output power level for signal transmission. antenna_capacitor - This selects the value of antenna tuning capacitor manually or automatically if set to zero.
- RDS related
- rds_ps_name - Sets the RDS ps name field for transmission.
- rds_radio_text - Sets the RDS radio text for transmission.
- rds_pi - Sets the RDS PI field for transmission.
- rds_pty - Sets the RDS PTY field for transmission.
- Region related
- preemphasis - sets the preemphasis to be applied for transmission.

4.26.4 RNL

This device also has an interface to measure received noise level. To do that, you should ioctl the device node. Here is an code of example:

```
int main (int argc, char *argv[])
{
        struct si4713_rnl rnl;
        int fd = open("/dev/radio0", O_RDWR);
        int rval;

        if (argc < 2)
                return -EINVAL;

        if (fd < 0)
                return fd;

        sscanf(argv[1], "%d", &rnl.frequency);

        rval = ioctl(fd, SI4713_IOC_MEASURE_RNL, &rnl);
        if (rval < 0)
                return rval;

        printf("received noise level: %d\n", rnl.rnl);

        close(fd);
}
```

The struct si4713_rnl and SI4713_IOC_MEASURE_RNL are defined under include/linux/platform_data/media/si4713.h.

4.26.5 Stereo/Mono and RDS subchannels

The device can also be configured using the available sub channels for transmission. To do that use S/G_MODULATOR ioctl and configure txsubchans properly. Refer to the V4L2 API specification for proper use of this ioctl.

4.26.6 Testing

Testing is usually done with v4l2-ctl utility for managing FM tuner cards. The tool can be found in v4l-dvb repository under v4l2-apps/util directory.

Example for setting rds ps name:

```
# v4l2-ctl -d /dev/radio0 --set-ctrl=rds_ps_name="Dummy"
```

4.27 The SI476x Driver

4.27.1 TODO for the driver

- According to the SiLabs' datasheet it is possible to update the firmware of the radio chip in the run-time, thus bringing it to the most recent version. Unfortunately I couldn't find any mentioning of the said firmware update for the old chips that I tested the driver against, so for chips like that the driver only exposes the old functionality.

4.27.2 Parameters exposed over debugfs

SI476x allow user to get multiple characteristics that can be very useful for EoL testing/RF performance estimation, parameters that have very little to do with V4L2 subsystem. Such parameters are exposed via debugfs and can be accessed via regular file I/O operations.

The drivers exposes following files:

- /sys/kernel/debug/<device-name>/acf This file contains ACF(Automatically Controlled Features) status information. The contents of the file is binary data of the following layout:

Offset	Name	Description
0x00	blend_int	Flag, set when stereo separation has crossed below the blend threshold
0x01	hblend_int	Flag, set when HiBlend cutoff frequency is lower than threshold
0x02	hicut_int	Flag, set when HiCut cutoff frequency is lower than threshold
0x03	chbw_int	Flag, set when channel filter bandwidth is less than threshold
0x04	softmute_int	Flag indicating that softmute attenuation has increased above softmute threshold
0x05	smute	0 - Audio is not soft muted 1 - Audio is soft muted
0x06	smattn	Soft mute attenuation level in dB
0x07	chbw	Channel filter bandwidth in kHz
0x08	hicut	HiCut cutoff frequency in units of 100Hz
0x09	hiblend	HiBlend cutoff frequency in units of 100 Hz
0x10	pilot	0 - Stereo pilot is not present 1 - Stereo pilot is present
0x11	stblend	Stereo blend in %

- /sys/kernel/debug/<device-name>/rds_blckcnt This file contains statistics about RDS receptions. It's binary data has the following layout:

Offset	Name	Description
0x00	expected	Number of expected RDS blocks
0x02	received	Number of received RDS blocks
0x04	uncorrectable	Number of uncorrectable RDS blocks

- /sys/kernel/debug/<device-name>/agc This file contains information about parameters pertaining to AGC(Automatic Gain Control)

  The layout is:

Offset	Name	Description
0x00	mxhi	0 - FM Mixer PD high threshold is not tripped 1 - FM Mixer PD high threshold is tripped
0x01	mxlo	ditto for FM Mixer PD low
0x02	lnahi	ditto for FM LNA PD high
0x03	lnalo	ditto for FM LNA PD low
0x04	fmagc1	FMAGC1 attenuator resistance (see datasheet for more detail)
0x05	fmagc2	ditto for FMAGC2
0x06	pgagain	PGA gain in dB
0x07	fmwblang	FM/WB LNA Gain in dB

- /sys/kernel/debug/<device-name>/rsq This file contains information about parameters pertaining to RSQ(Received Signal Quality)

  The layout is:

Offset	Name	Description
0x00	multhint	0 - multipath value has not crossed the Multipath high threshold 1 - multipath value has crossed the Multipath high threshold
0x01	multlint	ditto for Multipath low threshold
0x02	snrhint	0 - received signal's SNR has not crossed high threshold 1 - received signal's SNR has crossed high threshold
0x03	snrlint	ditto for low threshold
0x04	rssihint	ditto for RSSI high threshold
0x05	rssilint	ditto for RSSI low threshold
0x06	bltf	Flag indicating if seek command reached/wrapped seek band limit
0x07	snr_ready	Indicates that SNR metrics is ready
0x08	rssiready	ditto for RSSI metrics
0x09	injside	0 - Low-side injection is being used 1 - High-side injection is used
0x10	afcrl	Flag indicating if AFC rails
0x11	valid	Flag indicating if channel is valid
0x12	readfreq	Current tuned frequency
0x14	freqoff	Signed frequency offset in units of 2ppm
0x15	rssi	Signed value of RSSI in dBuV
0x16	snr	Signed RF SNR in dB
0x17	issi	Signed Image Strength Signal indicator
0x18	lassi	Signed Low side adjacent Channel Strength indicator
0x19	hassi	ditto fpr High side
0x20	mult	Multipath indicator
0x21	dev	Frequency deviation
0x24	assi	Adjacent channel SSI
0x25	usn	Ultrasonic noise indicator
0x26	pilotdev	Pilot deviation in units of 100 Hz
0x27	rdsdev	ditto for RDS
0x28	assidev	ditto for ASSI
0x29	strongdev	Frequency deviation
0x30	rdspi	RDS PI code

- /sys/kernel/debug/<device-name>/rsq_primary This file contains information about parameters pertaining to RSQ(Received Signal Quality) for primary tuner only. Layout is as the one above.

4.28 The Soc-Camera Drivers

Author: Guennadi Liakhovetski <g.liakhovetski@gmx.de>

4.28.1 Terminology

The following terms are used in this document:

- camera / camera device / camera sensor - a video-camera sensor chip, capable of connecting to a variety of systems and interfaces, typically uses i2c for control and configuration, and a parallel or a serial bus for data.
- camera host - an interface, to which a camera is connected. Typically a specialised interface, present on many SoCs, e.g. PXA27x and PXA3xx, SuperH, AVR32, i.MX27, i.MX31.
- camera host bus - a connection between a camera host and a camera. Can be parallel or serial, consists of data and control lines, e.g. clock, vertical and horizontal synchronization signals.

4.28.2 Purpose of the soc-camera subsystem

The soc-camera subsystem initially provided a unified API between camera host drivers and camera sensor drivers. Later the soc-camera sensor API has been replaced with the V4L2 standard subdev API. This also made camera driver re-use with non-soc-camera hosts possible. The camera host API to the soc-camera core has been preserved.

Soc-camera implements a V4L2 interface to the user, currently only the “mmap” method is supported by host drivers. However, the soc-camera core also provides support for the “read” method.

The subsystem has been designed to support multiple camera host interfaces and multiple cameras per interface, although most applications have only one camera sensor.

4.28.3 Existing drivers

As of 3.7 there are seven host drivers in the mainline: atmel-isi.c, mx1_camera.c (broken, scheduled for removal), mx2_camera.c, mx3_camera.c, omap1_camera.c, pxa_camera.c, sh_mobile_ceu_camera.c, and multiple sensor drivers under drivers/media/i2c/soc_camera/.

4.28.4 Camera host API

A host camera driver is registered using the

```
soc_camera_host_register(struct soc_camera_host *);
```

function. The host object can be initialized as follows:

```
struct soc_camera_host  *ici;
ici->drv_name           = DRV_NAME;
ici->ops                = &camera_host_ops;
ici->priv               = pcdev;
ici->v4l2_dev.dev       = &pdev->dev;
ici->nr                 = pdev->id;
```

All camera host methods are passed in a struct soc_camera_host_ops:

```
static struct soc_camera_host_ops camera_host_ops = {
        .owner          = THIS_MODULE,
        .add            = camera_add_device,
        .remove         = camera_remove_device,
```

```
        .set_fmt        = camera_set_fmt_cap,
        .try_fmt        = camera_try_fmt_cap,
        .init_videobuf2 = camera_init_videobuf2,
        .poll           = camera_poll,
        .querycap       = camera_querycap,
        .set_bus_param  = camera_set_bus_param,
        /* The rest of host operations are optional */
};
```

.add and .remove methods are called when a sensor is attached to or detached from the host. .set_bus_param is used to configure physical connection parameters between the host and the sensor. .init_videobuf2 is called by soc-camera core when a video-device is opened, the host driver would typically call vb2_queue_init() in this method. Further video-buffer management is implemented completely by the specific camera host driver. If the host driver supports non-standard pixel format conversion, it should implement a .get_formats and, possibly, a .put_formats operations. See below for more details about format conversion. The rest of the methods are called from respective V4L2 operations.

4.28.5 Camera API

Sensor drivers can use struct soc_camera_link, typically provided by the platform, and used to specify to which camera host bus the sensor is connected, and optionally provide platform .power and .reset methods for the camera. This struct is provided to the camera driver via the I2C client device platform data and can be obtained, using the soc_camera_i2c_to_link() macro. Care should be taken, when using soc_camera_vdev_to_subdev() and when accessing struct soc_camera_device, using v4l2_get_subdev_hostdata(): both only work, when running on an soc-camera host. The actual camera driver operation is implemented using the V4L2 subdev API. Additionally soc-camera camera drivers can use auxiliary soc-camera helper functions like soc_camera_power_on() and soc_camera_power_off(), which switch regulators, provided by the platform and call board-specific power switching methods. soc_camera_apply_board_flags() takes camera bus configuration capability flags and applies any board transformations, e.g. signal polarity inversion. soc_mbus_get_fmtdesc() can be used to obtain a pixel format descriptor, corresponding to a certain media-bus pixel format code. soc_camera_limit_side() can be used to restrict beginning and length of a frame side, based on camera capabilities.

4.28.6 VIDIOC_S_CROP and VIDIOC_S_FMT behaviour

Above user ioctls modify image geometry as follows:

VIDIOC_S_CROP: sets location and sizes of the sensor window. Unit is one sensor pixel. Changing sensor window sizes preserves any scaling factors, therefore user window sizes change as well.

VIDIOC_S_FMT: sets user window. Should preserve previously set sensor window as much as possible by modifying scaling factors. If the sensor window cannot be preserved precisely, it may be changed too.

In soc-camera there are two locations, where scaling and cropping can take place: in the camera driver and in the host driver. User ioctls are first passed to the host driver, which then generally passes them down to the camera driver. It is more efficient to perform scaling and cropping in the camera driver to save camera bus bandwidth and maximise the framerate. However, if the camera driver failed to set the required parameters with sufficient precision, the host driver may decide to also use its own scaling and cropping to fulfill the user's request.

Camera drivers are interfaced to the soc-camera core and to host drivers over the v4l2-subdev API, which is completely functional, it doesn't pass any data. Therefore all camera drivers shall reply to .g_fmt() requests with their current output geometry. This is necessary to correctly configure the camera bus. .s_fmt() and .try_fmt() have to be implemented too. Sensor window and scaling factors have to be maintained by camera drivers internally. According to the V4L2 API all capture drivers must support the VIDIOC_CROPCAP ioctl, hence we rely on camera drivers implementing .cropcap(). If the camera driver does not support cropping, it may choose to not implement .s_crop(), but to enable cropping support by the camera host driver at least the .g_crop method must be implemented.

User window geometry is kept in .user_width and .user_height fields in struct soc_camera_device and used by the soc-camera core and host drivers. The core updates these fields upon successful completion of a .s_fmt() call, but if these fields change elsewhere, e.g. during .s_crop() processing, the host driver is responsible for updating them.

4.28.7 Format conversion

V4L2 distinguishes between pixel formats, as they are stored in memory, and as they are transferred over a media bus. Soc-camera provides support to conveniently manage these formats. A table of standard transformations is maintained by soc-camera core, which describes, what FOURCC pixel format will be obtained, if a media-bus pixel format is stored in memory according to certain rules. E.g. if MEDIA_BUS_FMT_YUYV8_2X8 data is sampled with 8 bits per sample and stored in memory in the little-endian order with no gaps between bytes, data in memory will represent the V4L2_PIX_FMT_YUYV FOURCC format. These standard transformations will be used by soc-camera or by camera host drivers to configure camera drivers to produce the FOURCC format, requested by the user, using the VIDIOC_S_FMT ioctl(). Apart from those standard format conversions, host drivers can also provide their own conversion rules by implementing a .get_formats and, if required, a .put_formats methods.

4.29 The Linux USB Video Class (UVC) driver

This file documents some driver-specific aspects of the UVC driver, such as driver-specific ioctls and implementation notes.

Questions and remarks can be sent to the Linux UVC development mailing list at linux-uvc-devel@lists.berlios.de.

4.29.1 Extension Unit (XU) support

Introduction

The UVC specification allows for vendor-specific extensions through extension units (XUs). The Linux UVC driver supports extension unit controls (XU controls) through two separate mechanisms:

- through mappings of XU controls to V4L2 controls
- through a driver-specific ioctl interface

The first one allows generic V4L2 applications to use XU controls by mapping certain XU controls onto V4L2 controls, which then show up during ordinary control enumeration.

The second mechanism requires uvcvideo-specific knowledge for the application to access XU controls but exposes the entire UVC XU concept to user space for maximum flexibility.

Both mechanisms complement each other and are described in more detail below.

Control mappings

The UVC driver provides an API for user space applications to define so-called control mappings at runtime. These allow for individual XU controls or byte ranges thereof to be mapped to new V4L2 controls. Such controls appear and function exactly like normal V4L2 controls (i.e. the stock controls, such as brightness, contrast, etc.). However, reading or writing of such a V4L2 controls triggers a read or write of the associated XU control.

The ioctl used to create these control mappings is called UVCIOC_CTRL_MAP. Previous driver versions (before 0.2.0) required another ioctl to be used beforehand (UVCIOC_CTRL_ADD) to pass XU control information to the UVC driver. This is no longer necessary as newer uvcvideo versions query the information directly from the device.

For details on the UVCIOC_CTRL_MAP ioctl please refer to the section titled “IOCTL reference” below.

3. Driver specific XU control interface

For applications that need to access XU controls directly, e.g. for testing purposes, firmware upload, or accessing binary controls, a second mechanism to access XU controls is provided in the form of a driver-specific ioctl, namely UVCIOC_CTRL_QUERY.

A call to this ioctl allows applications to send queries to the UVC driver that directly map to the low-level UVC control requests.

In order to make such a request the UVC unit ID of the control’s extension unit and the control selector need to be known. This information either needs to be hardcoded in the application or queried using other ways such as by parsing the UVC descriptor or, if available, using the media controller API to enumerate a device’s entities.

Unless the control size is already known it is necessary to first make a UVC_GET_LEN requests in order to be able to allocate a sufficiently large buffer and set the buffer size to the correct value. Similarly, to find out whether UVC_GET_CUR or UVC_SET_CUR are valid requests for a given control, a UVC_GET_INFO request should be made. The bits 0 (GET supported) and 1 (SET supported) of the resulting byte indicate which requests are valid.

With the addition of the UVCIOC_CTRL_QUERY ioctl the UVCIOC_CTRL_GET and UVCIOC_CTRL_SET ioctls have become obsolete since their functionality is a subset of the former ioctl. For the time being they are still supported but application developers are encouraged to use UVCIOC_CTRL_QUERY instead.

For details on the UVCIOC_CTRL_QUERY ioctl please refer to the section titled “IOCTL reference” below.

Security

The API doesn’t currently provide a fine-grained access control facility. The UVCIOC_CTRL_ADD and UVCIOC_CTRL_MAP ioctls require super user permissions.

Suggestions on how to improve this are welcome.

Debugging

In order to debug problems related to XU controls or controls in general it is recommended to enable the UVC_TRACE_CONTROL bit in the module parameter ‘trace’. This causes extra output to be written into the system log.

IOCTL reference

UVCIOC_CTRL_MAP - Map a UVC control to a V4L2 control

Argument: struct uvc_xu_control_mapping

Description:

This ioctl creates a mapping between a UVC control or part of a UVC control and a V4L2 control. Once mappings are defined, userspace applications can access vendor-defined UVC control through the V4L2 control API.

To create a mapping, applications fill the uvc_xu_control_mapping structure with information about an existing UVC control defined with UVCIOC_CTRL_ADD and a new V4L2 control.

A UVC control can be mapped to several V4L2 controls. For instance, a UVC pan/tilt control could be mapped to separate pan and tilt V4L2 controls. The UVC control is divided into non overlapping fields using the ‘size’ and ‘offset’ fields and are then independently mapped to V4L2 control.

For signed integer V4L2 controls the data_type field should be set to UVC_CTRL_DATA_TYPE_SIGNED. Other values are currently ignored.

Return value:

On success 0 is returned. On error -1 is returned and errno is set appropriately.

ENOMEM Not enough memory to perform the operation.

EPERM Insufficient privileges (super user privileges are required).

EINVAL No such UVC control.

EOVERFLOW The requested offset and size would overflow the UVC control.

EEXIST Mapping already exists.

Data types:

```
* struct uvc_xu_control_mapping

__u32   id              V4L2 control identifier
__u8    name[32]        V4L2 control name
__u8    entity[16]      UVC extension unit GUID
__u8    selector        UVC control selector
__u8    size            V4L2 control size (in bits)
__u8    offset          V4L2 control offset (in bits)
enum v4l2_ctrl_type
        v4l2_type       V4L2 control type
enum uvc_control_data_type
        data_type       UVC control data type
struct uvc_menu_info
        *menu_info      Array of menu entries (for menu controls only)
__u32   menu_count      Number of menu entries (for menu controls only)

* struct uvc_menu_info

__u32   value           Menu entry value used by the device
__u8    name[32]        Menu entry name


* enum uvc_control_data_type

UVC_CTRL_DATA_TYPE_RAW          Raw control (byte array)
UVC_CTRL_DATA_TYPE_SIGNED       Signed integer
UVC_CTRL_DATA_TYPE_UNSIGNED     Unsigned integer
UVC_CTRL_DATA_TYPE_BOOLEAN      Boolean
UVC_CTRL_DATA_TYPE_ENUM         Enumeration
UVC_CTRL_DATA_TYPE_BITMASK      Bitmask
```

UVCIOC_CTRL_QUERY - Query a UVC XU control

Argument: struct uvc_xu_control_query

Description:

This ioctl queries a UVC XU control identified by its extension unit ID and control selector.

There are a number of different queries available that closely correspond to the low-level control requests described in the UVC specification. These requests are:

UVC_GET_CUR Obtain the current value of the control.

UVC_GET_MIN Obtain the minimum value of the control.

UVC_GET_MAX Obtain the maximum value of the control.

UVC_GET_DEF Obtain the default value of the control.

UVC_GET_RES Query the resolution of the control, i.e. the step size of the allowed control values.

UVC_GET_LEN Query the size of the control in bytes.

UVC_GET_INFO Query the control information bitmap, which indicates whether get/set requests are supported.

UVC_SET_CUR Update the value of the control.

Applications must set the 'size' field to the correct length for the control. Exceptions are the UVC_GET_LEN and UVC_GET_INFO queries, for which the size must be set to 2 and 1, respectively. The 'data' field must point to a valid writable buffer big enough to hold the indicated number of data bytes.

Data is copied directly from the device without any driver-side processing. Applications are responsible for data buffer formatting, including little-endian/big-endian conversion. This is particularly important for the result of the UVC_GET_LEN requests, which is always returned as a little-endian 16-bit integer by the device.

Return value:

On success 0 is returned. On error -1 is returned and errno is set appropriately.

ENOENT The device does not support the given control or the specified extension unit could not be found.

ENOBUFS The specified buffer size is incorrect (too big or too small).

EINVAL An invalid request code was passed.

EBADRQC The given request is not supported by the given control.

EFAULT The data pointer references an inaccessible memory area.

Data types:

```
* struct uvc_xu_control_query

__u8    unit            Extension unit ID
__u8    selector        Control selector
__u8    query           Request code to send to the device
__u16   size            Control data size (in bytes)
__u8    *data           Control value
```

4.30 The Virtual Video Test Driver (vivid)

This driver emulates video4linux hardware of various types: video capture, video output, vbi capture and output, radio receivers and transmitters and a software defined radio receiver. In addition a simple framebuffer device is available for testing capture and output overlays.

Up to 64 vivid instances can be created, each with up to 16 inputs and 16 outputs.

Each input can be a webcam, TV capture device, S-Video capture device or an HDMI capture device. Each output can be an S-Video output device or an HDMI output device.

These inputs and outputs act exactly as a real hardware device would behave. This allows you to use this driver as a test input for application development, since you can test the various features without requiring special hardware.

This document describes the features implemented by this driver:

- Support for read()/write(), MMAP, USERPTR and DMABUF streaming I/O.
- A large list of test patterns and variations thereof

- Working brightness, contrast, saturation and hue controls
- Support for the alpha color component
- Full colorspace support, including limited/full RGB range
- All possible control types are present
- Support for various pixel aspect ratios and video aspect ratios
- Error injection to test what happens if errors occur
- Supports crop/compose/scale in any combination for both input and output
- Can emulate up to 4K resolutions
- All Field settings are supported for testing interlaced capturing
- Supports all standard YUV and RGB formats, including two multiplanar YUV formats
- Raw and Sliced VBI capture and output support
- Radio receiver and transmitter support, including RDS support
- Software defined radio (SDR) support
- Capture and output overlay support

These features will be described in more detail below.

4.30.1 Configuring the driver

By default the driver will create a single instance that has a video capture device with webcam, TV, S-Video and HDMI inputs, a video output device with S-Video and HDMI outputs, one vbi capture device, one vbi output device, one radio receiver device, one radio transmitter device and one SDR device.

The number of instances, devices, video inputs and outputs and their types are all configurable using the following module options:

- n_devs:

  number of driver instances to create. By default set to 1. Up to 64 instances can be created.

- node_types:

  which devices should each driver instance create. An array of hexadecimal values, one for each instance. The default is 0x1d3d. Each value is a bitmask with the following meaning:

  - bit 0: Video Capture node
  - bit 2-3: VBI Capture node: 0 = none, 1 = raw vbi, 2 = sliced vbi, 3 = both
  - bit 4: Radio Receiver node
  - bit 5: Software Defined Radio Receiver node
  - bit 8: Video Output node
  - bit 10-11: VBI Output node: 0 = none, 1 = raw vbi, 2 = sliced vbi, 3 = both
  - bit 12: Radio Transmitter node
  - bit 16: Framebuffer for testing overlays

  So to create four instances, the first two with just one video capture device, the second two with just one video output device you would pass these module options to vivid:

  ```
  n_devs=4 node_types=0x1,0x1,0x100,0x100
  ```

- num_inputs:

the number of inputs, one for each instance. By default 4 inputs are created for each video capture device. At most 16 inputs can be created, and there must be at least one.

- input_types:

  the input types for each instance, the default is 0xe4. This defines what the type of each input is when the inputs are created for each driver instance. This is a hexadecimal value with up to 16 pairs of bits, each pair gives the type and bits 0-1 map to input 0, bits 2-3 map to input 1, 30-31 map to input 15. Each pair of bits has the following meaning:

  - 00: this is a webcam input
  - 01: this is a TV tuner input
  - 10: this is an S-Video input
  - 11: this is an HDMI input

  So to create a video capture device with 8 inputs where input 0 is a TV tuner, inputs 1-3 are S-Video inputs and inputs 4-7 are HDMI inputs you would use the following module options:

```
num_inputs=8 input_types=0xffa9
```

- num_outputs:

  the number of outputs, one for each instance. By default 2 outputs are created for each video output device. At most 16 outputs can be created, and there must be at least one.

- output_types:

  the output types for each instance, the default is 0x02. This defines what the type of each output is when the outputs are created for each driver instance. This is a hexadecimal value with up to 16 bits, each bit gives the type and bit 0 maps to output 0, bit 1 maps to output 1, bit 15 maps to output 15. The meaning of each bit is as follows:

  - 0: this is an S-Video output
  - 1: this is an HDMI output

  So to create a video output device with 8 outputs where outputs 0-3 are S-Video outputs and outputs 4-7 are HDMI outputs you would use the following module options:

```
num_outputs=8 output_types=0xf0
```

- vid_cap_nr:

  give the desired videoX start number for each video capture device. The default is -1 which will just take the first free number. This allows you to map capture video nodes to specific videoX device nodes. Example:

```
n_devs=4 vid_cap_nr=2,4,6,8
```

  This will attempt to assign /dev/video2 for the video capture device of the first vivid instance, video4 for the next up to video8 for the last instance. If it can't succeed, then it will just take the next free number.

- vid_out_nr:

  give the desired videoX start number for each video output device. The default is -1 which will just take the first free number.

- vbi_cap_nr:

  give the desired vbiX start number for each vbi capture device. The default is -1 which will just take the first free number.

- vbi_out_nr:

give the desired vbiX start number for each vbi output device. The default is -1 which will just take the first free number.

- radio_rx_nr:

  give the desired radioX start number for each radio receiver device. The default is -1 which will just take the first free number.

- radio_tx_nr:

  give the desired radioX start number for each radio transmitter device. The default is -1 which will just take the first free number.

- sdr_cap_nr:

  give the desired swradioX start number for each SDR capture device. The default is -1 which will just take the first free number.

- ccs_cap_mode:

  specify the allowed video capture crop/compose/scaling combination for each driver instance. Video capture devices can have any combination of cropping, composing and scaling capabilities and this will tell the vivid driver which of those is should emulate. By default the user can select this through controls.

  The value is either -1 (controlled by the user) or a set of three bits, each enabling (1) or disabling (0) one of the features:

  - bit 0:

    Enable crop support. Cropping will take only part of the incoming picture.

  - bit 1:

    Enable compose support. Composing will copy the incoming picture into a larger buffer.

  - bit 2:

    Enable scaling support. Scaling can scale the incoming picture. The scaler of the vivid driver can enlarge up or down to four times the original size. The scaler is very simple and low-quality. Simplicity and speed were key, not quality.

  Note that this value is ignored by webcam inputs: those enumerate discrete framesizes and that is incompatible with cropping, composing or scaling.

- ccs_out_mode:

  specify the allowed video output crop/compose/scaling combination for each driver instance. Video output devices can have any combination of cropping, composing and scaling capabilities and this will tell the vivid driver which of those is should emulate. By default the user can select this through controls.

  The value is either -1 (controlled by the user) or a set of three bits, each enabling (1) or disabling (0) one of the features:

  - bit 0:

    Enable crop support. Cropping will take only part of the outgoing buffer.

  - bit 1:

    Enable compose support. Composing will copy the incoming buffer into a larger picture frame.

  - bit 2:

    Enable scaling support. Scaling can scale the incoming buffer. The scaler of the vivid driver can enlarge up or down to four times the original size. The scaler is very simple and low-quality. Simplicity and speed were key, not quality.

- multiplanar:

  select whether each device instance supports multi-planar formats, and thus the V4L2 multi-planar API. By default device instances are single-planar.

  This module option can override that for each instance. Values are:

  - 1: this is a single-planar instance.
  - 2: this is a multi-planar instance.

- vivid_debug:

  enable driver debugging info

- no_error_inj:

  if set disable the error injecting controls. This option is needed in order to run a tool like v4l2-compliance. Tools like that exercise all controls including a control like 'Disconnect' which emulates a USB disconnect, making the device inaccessible and so all tests that v4l2-compliance is doing will fail afterwards.

  There may be other situations as well where you want to disable the error injection support of vivid. When this option is set, then the controls that select crop, compose and scale behavior are also removed. Unless overridden by ccs_cap_mode and/or ccs_out_mode the will default to enabling crop, compose and scaling.

Taken together, all these module options allow you to precisely customize the driver behavior and test your application with all sorts of permutations. It is also very suitable to emulate hardware that is not yet available, e.g. when developing software for a new upcoming device.

4.30.2 Video Capture

This is probably the most frequently used feature. The video capture device can be configured by using the module options num_inputs, input_types and ccs_cap_mode (see section 1 for more detailed information), but by default four inputs are configured: a webcam, a TV tuner, an S-Video and an HDMI input, one input for each input type. Those are described in more detail below.

Special attention has been given to the rate at which new frames become available. The jitter will be around 1 jiffie (that depends on the HZ configuration of your kernel, so usually 1/100, 1/250 or 1/1000 of a second), but the long-term behavior is exactly following the framerate. So a framerate of 59.94 Hz is really different from 60 Hz. If the framerate exceeds your kernel's HZ value, then you will get dropped frames, but the frame/field sequence counting will keep track of that so the sequence count will skip whenever frames are dropped.

Webcam Input

The webcam input supports three framesizes: 320x180, 640x360 and 1280x720. It supports frames per second settings of 10, 15, 25, 30, 50 and 60 fps. Which ones are available depends on the chosen framesize: the larger the framesize, the lower the maximum frames per second.

The initially selected colorspace when you switch to the webcam input will be sRGB.

TV and S-Video Inputs

The only difference between the TV and S-Video input is that the TV has a tuner. Otherwise they behave identically.

These inputs support audio inputs as well: one TV and one Line-In. They both support all TV standards. If the standard is queried, then the Vivid controls 'Standard Signal Mode' and 'Standard' determine what the result will be.

These inputs support all combinations of the field setting. Special care has been taken to faithfully reproduce how fields are handled for the different TV standards. This is particularly noticeable when generating a horizontally moving image so the temporal effect of using interlaced formats becomes clearly visible. For 50 Hz standards the top field is the oldest and the bottom field is the newest in time. For 60 Hz standards that is reversed: the bottom field is the oldest and the top field is the newest in time.

When you start capturing in V4L2_FIELD_ALTERNATE mode the first buffer will contain the top field for 50 Hz standards and the bottom field for 60 Hz standards. This is what capture hardware does as well.

Finally, for PAL/SECAM standards the first half of the top line contains noise. This simulates the Wide Screen Signal that is commonly placed there.

The initially selected colorspace when you switch to the TV or S-Video input will be SMPTE-170M.

The pixel aspect ratio will depend on the TV standard. The video aspect ratio can be selected through the 'Standard Aspect Ratio' Vivid control. Choices are '4x3', '16x9' which will give letterboxed widescreen video and '16x9 Anamorphic' which will give full screen squashed anamorphic widescreen video that will need to be scaled accordingly.

The TV 'tuner' supports a frequency range of 44-958 MHz. Channels are available every 6 MHz, starting from 49.25 MHz. For each channel the generated image will be in color for the +/- 0.25 MHz around it, and in grayscale for +/- 1 MHz around the channel. Beyond that it is just noise. The VIDIOC_G_TUNER ioctl will return 100% signal strength for +/- 0.25 MHz and 50% for +/- 1 MHz. It will also return correct afc values to show whether the frequency is too low or too high.

The audio subchannels that are returned are MONO for the +/- 1 MHz range around a valid channel frequency. When the frequency is within +/- 0.25 MHz of the channel it will return either MONO, STEREO, either MONO | SAP (for NTSC) or LANG1 | LANG2 (for others), or STEREO | SAP.

Which one is returned depends on the chosen channel, each next valid channel will cycle through the possible audio subchannel combinations. This allows you to test the various combinations by just switching channels..

Finally, for these inputs the v4l2_timecode struct is filled in in the dequeued v4l2_buffer struct.

HDMI Input

The HDMI inputs supports all CEA-861 and DMT timings, both progressive and interlaced, for pixelclock frequencies between 25 and 600 MHz. The field mode for interlaced formats is always V4L2_FIELD_ALTERNATE. For HDMI the field order is always top field first, and when you start capturing an interlaced format you will receive the top field first.

The initially selected colorspace when you switch to the HDMI input or select an HDMI timing is based on the format resolution: for resolutions less than or equal to 720x576 the colorspace is set to SMPTE-170M, for others it is set to REC-709 (CEA-861 timings) or sRGB (VESA DMT timings).

The pixel aspect ratio will depend on the HDMI timing: for 720x480 is it set as for the NTSC TV standard, for 720x576 it is set as for the PAL TV standard, and for all others a 1:1 pixel aspect ratio is returned.

The video aspect ratio can be selected through the 'DV Timings Aspect Ratio' Vivid control. Choices are 'Source Width x Height' (just use the same ratio as the chosen format), '4x3' or '16x9', either of which can result in pillarboxed or letterboxed video.

For HDMI inputs it is possible to set the EDID. By default a simple EDID is provided. You can only set the EDID for HDMI inputs. Internally, however, the EDID is shared between all HDMI inputs.

No interpretation is done of the EDID data with the exception of the physical address. See the CEC section for more details.

There is a maximum of 15 HDMI inputs (if there are more, then they will be reduced to 15) since that's the limitation of the EDID physical address.

4.30.3 Video Output

The video output device can be configured by using the module options num_outputs, output_types and ccs_out_mode (see section 1 for more detailed information), but by default two outputs are configured: an S-Video and an HDMI input, one output for each output type. Those are described in more detail below.

Like with video capture the framerate is also exact in the long term.

S-Video Output

This output supports audio outputs as well: "Line-Out 1" and "Line-Out 2". The S-Video output supports all TV standards.

This output supports all combinations of the field setting.

The initially selected colorspace when you switch to the TV or S-Video input will be SMPTE-170M.

HDMI Output

The HDMI output supports all CEA-861 and DMT timings, both progressive and interlaced, for pixelclock frequencies between 25 and 600 MHz. The field mode for interlaced formats is always V4L2_FIELD_ALTERNATE.

The initially selected colorspace when you switch to the HDMI output or select an HDMI timing is based on the format resolution: for resolutions less than or equal to 720x576 the colorspace is set to SMPTE-170M, for others it is set to REC-709 (CEA-861 timings) or sRGB (VESA DMT timings).

The pixel aspect ratio will depend on the HDMI timing: for 720x480 is it set as for the NTSC TV standard, for 720x576 it is set as for the PAL TV standard, and for all others a 1:1 pixel aspect ratio is returned.

An HDMI output has a valid EDID which can be obtained through VIDIOC_G_EDID.

There is a maximum of 15 HDMI outputs (if there are more, then they will be reduced to 15) since that's the limitation of the EDID physical address. See also the CEC section for more details.

4.30.4 VBI Capture

There are three types of VBI capture devices: those that only support raw (undecoded) VBI, those that only support sliced (decoded) VBI and those that support both. This is determined by the node_types module option. In all cases the driver will generate valid VBI data: for 60 Hz standards it will generate Closed Caption and XDS data. The closed caption stream will alternate between "Hello world!" and "Closed captions test" every second. The XDS stream will give the current time once a minute. For 50 Hz standards it will generate the Wide Screen Signal which is based on the actual Video Aspect Ratio control setting and teletext pages 100-159, one page per frame.

The VBI device will only work for the S-Video and TV inputs, it will give back an error if the current input is a webcam or HDMI.

4.30.5 VBI Output

There are three types of VBI output devices: those that only support raw (undecoded) VBI, those that only support sliced (decoded) VBI and those that support both. This is determined by the node_types module option.

The sliced VBI output supports the Wide Screen Signal and the teletext signal for 50 Hz standards and Closed Captioning + XDS for 60 Hz standards.

The VBI device will only work for the S-Video output, it will give back an error if the current output is HDMI.

4.30.6 Radio Receiver

The radio receiver emulates an FM/AM/SW receiver. The FM band also supports RDS. The frequency ranges are:

- FM: 64 MHz - 108 MHz
- AM: 520 kHz - 1710 kHz
- SW: 2300 kHz - 26.1 MHz

Valid channels are emulated every 1 MHz for FM and every 100 kHz for AM and SW. The signal strength decreases the further the frequency is from the valid frequency until it becomes 0% at +/- 50 kHz (FM) or 5 kHz (AM/SW) from the ideal frequency. The initial frequency when the driver is loaded is set to 95 MHz.

The FM receiver supports RDS as well, both using 'Block I/O' and 'Controls' modes. In the 'Controls' mode the RDS information is stored in read-only controls. These controls are updated every time the frequency is changed, or when the tuner status is requested. The Block I/O method uses the read() interface to pass the RDS blocks on to the application for decoding.

The RDS signal is 'detected' for +/- 12.5 kHz around the channel frequency, and the further the frequency is away from the valid frequency the more RDS errors are randomly introduced into the block I/O stream, up to 50% of all blocks if you are +/- 12.5 kHz from the channel frequency. All four errors can occur in equal proportions: blocks marked 'CORRECTED', blocks marked 'ERROR', blocks marked 'INVALID' and dropped blocks.

The generated RDS stream contains all the standard fields contained in a 0B group, and also radio text and the current time.

The receiver supports HW frequency seek, either in Bounded mode, Wrap Around mode or both, which is configurable with the "Radio HW Seek Mode" control.

4.30.7 Radio Transmitter

The radio transmitter emulates an FM/AM/SW transmitter. The FM band also supports RDS. The frequency ranges are:

- FM: 64 MHz - 108 MHz
- AM: 520 kHz - 1710 kHz
- SW: 2300 kHz - 26.1 MHz

The initial frequency when the driver is loaded is 95.5 MHz.

The FM transmitter supports RDS as well, both using 'Block I/O' and 'Controls' modes. In the 'Controls' mode the transmitted RDS information is configured using controls, and in 'Block I/O' mode the blocks are passed to the driver using write().

4.30.8 Software Defined Radio Receiver

The SDR receiver has three frequency bands for the ADC tuner:

- 300 kHz
- 900 kHz - 2800 kHz
- 3200 kHz

The RF tuner supports 50 MHz - 2000 MHz.

The generated data contains the In-phase and Quadrature components of a 1 kHz tone that has an amplitude of sqrt(2).

4.30.9 Controls

Different devices support different controls. The sections below will describe each control and which devices support them.

User Controls - Test Controls

The Button, Boolean, Integer 32 Bits, Integer 64 Bits, Menu, String, Bitmask and Integer Menu are controls that represent all possible control types. The Menu control and the Integer Menu control both have 'holes' in their menu list, meaning that one or more menu items return EINVAL when VIDIOC_QUERYMENU is called. Both menu controls also have a non-zero minimum control value. These features allow you to check if your application can handle such things correctly. These controls are supported for every device type.

User Controls - Video Capture

The following controls are specific to video capture.

The Brightness, Contrast, Saturation and Hue controls actually work and are standard. There is one special feature with the Brightness control: each video input has its own brightness value, so changing input will restore the brightness for that input. In addition, each video input uses a different brightness range (minimum and maximum control values). Switching inputs will cause a control event to be sent with the V4L2_EVENT_CTRL_CH_RANGE flag set. This allows you to test controls that can change their range.

The 'Gain, Automatic' and Gain controls can be used to test volatile controls: if 'Gain, Automatic' is set, then the Gain control is volatile and changes constantly. If 'Gain, Automatic' is cleared, then the Gain control is a normal control.

The 'Horizontal Flip' and 'Vertical Flip' controls can be used to flip the image. These combine with the 'Sensor Flipped Horizontally/Vertically' Vivid controls.

The 'Alpha Component' control can be used to set the alpha component for formats containing an alpha channel.

User Controls - Audio

The following controls are specific to video capture and output and radio receivers and transmitters.

The 'Volume' and 'Mute' audio controls are typical for such devices to control the volume and mute the audio. They don't actually do anything in the vivid driver.

Vivid Controls

These vivid custom controls control the image generation, error injection, etc.

Test Pattern Controls

The Test Pattern Controls are all specific to video capture.

- Test Pattern:

  selects which test pattern to use. Use the CSC Colorbar for testing colorspace conversions: the colors used in that test pattern map to valid colors in all colorspaces. The colorspace conversion is disabled for the other test patterns.

- OSD Text Mode:

  selects whether the text superimposed on the test pattern should be shown, and if so, whether only counters should be displayed or the full text.

- Horizontal Movement:

  selects whether the test pattern should move to the left or right and at what speed.

- Vertical Movement:

  does the same for the vertical direction.

- Show Border:

  show a two-pixel wide border at the edge of the actual image, excluding letter or pillarboxing.

- Show Square:

  show a square in the middle of the image. If the image is displayed with the correct pixel and image aspect ratio corrections, then the width and height of the square on the monitor should be the same.

- Insert SAV Code in Image:

  adds a SAV (Start of Active Video) code to the image. This can be used to check if such codes in the image are inadvertently interpreted instead of being ignored.

- Insert EAV Code in Image:

  does the same for the EAV (End of Active Video) code.

Capture Feature Selection Controls

These controls are all specific to video capture.

- Sensor Flipped Horizontally:

  the image is flipped horizontally and the V4L2_IN_ST_HFLIP input status flag is set. This emulates the case where a sensor is for example mounted upside down.

- Sensor Flipped Vertically:

  the image is flipped vertically and the V4L2_IN_ST_VFLIP input status flag is set. This emulates the case where a sensor is for example mounted upside down.

- Standard Aspect Ratio:

  selects if the image aspect ratio as used for the TV or S-Video input should be 4x3, 16x9 or anamorphic widescreen. This may introduce letterboxing.

- DV Timings Aspect Ratio:

  selects if the image aspect ratio as used for the HDMI input should be the same as the source width and height ratio, or if it should be 4x3 or 16x9. This may introduce letter or pillarboxing.

- Timestamp Source:

  selects when the timestamp for each buffer is taken.

- Colorspace:

  selects which colorspace should be used when generating the image. This only applies if the CSC Colorbar test pattern is selected, otherwise the test pattern will go through unconverted. This behavior is also what you want, since a 75% Colorbar should really have 75% signal intensity and should not be affected by colorspace conversions.

  Changing the colorspace will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a detected colorspace change.

- Transfer Function:

selects which colorspace transfer function should be used when generating an image. This only applies if the CSC Colorbar test pattern is selected, otherwise the test pattern will go through unconverted. This behavior is also what you want, since a 75% Colorbar should really have 75% signal intensity and should not be affected by colorspace conversions.

Changing the transfer function will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a detected colorspace change.

- Y'CbCr Encoding:

  selects which Y'CbCr encoding should be used when generating a Y'CbCr image. This only applies if the format is set to a Y'CbCr format as opposed to an RGB format.

  Changing the Y'CbCr encoding will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a detected colorspace change.

- Quantization:

  selects which quantization should be used for the RGB or Y'CbCr encoding when generating the test pattern.

  Changing the quantization will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a detected colorspace change.

- Limited RGB Range (16-235):

  selects if the RGB range of the HDMI source should be limited or full range. This combines with the Digital Video 'Rx RGB Quantization Range' control and can be used to test what happens if a source provides you with the wrong quantization range information. See the description of that control for more details.

- Apply Alpha To Red Only:

  apply the alpha channel as set by the 'Alpha Component' user control to the red color of the test pattern only.

- Enable Capture Cropping:

  enables crop support. This control is only present if the ccs_cap_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Enable Capture Composing:

  enables composing support. This control is only present if the ccs_cap_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Enable Capture Scaler:

  enables support for a scaler (maximum 4 times upscaling and downscaling). This control is only present if the ccs_cap_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Maximum EDID Blocks:

  determines how many EDID blocks the driver supports. Note that the vivid driver does not actually interpret new EDID data, it just stores it. It allows for up to 256 EDID blocks which is the maximum supported by the standard.

- Fill Percentage of Frame:

  can be used to draw only the top X percent of the image. Since each frame has to be drawn by the driver, this demands a lot of the CPU. For large resolutions this becomes problematic. By drawing only part of the image this CPU load can be reduced.

Output Feature Selection Controls

These controls are all specific to video output.

- Enable Output Cropping:

  enables crop support. This control is only present if the ccs_out_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Enable Output Composing:

  enables composing support. This control is only present if the ccs_out_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

- Enable Output Scaler:

  enables support for a scaler (maximum 4 times upscaling and downscaling). This control is only present if the ccs_out_mode module option is set to the default value of -1 and if the no_error_inj module option is set to 0 (the default).

Error Injection Controls

The following two controls are only valid for video and vbi capture.

- Standard Signal Mode:

  selects the behavior of VIDIOC_QUERYSTD: what should it return?

  Changing this control will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a changed input condition (e.g. a cable was plugged in or out).

- Standard:

  selects the standard that VIDIOC_QUERYSTD should return if the previous control is set to “Selected Standard”.

  Changing this control will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a changed input standard.

The following two controls are only valid for video capture.

- **DV Timings Signal Mode:** selects the behavior of VIDIOC_QUERY_DV_TIMINGS: what should it return?

  Changing this control will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates a changed input condition (e.g. a cable was plugged in or out).

- DV Timings:

  selects the timings the VIDIOC_QUERY_DV_TIMINGS should return if the previous control is set to “Selected DV Timings”.

  Changing this control will result in the V4L2_EVENT_SOURCE_CHANGE to be sent since it emulates changed input timings.

The following controls are only present if the no_error_inj module option is set to 0 (the default). These controls are valid for video and vbi capture and output streams and for the SDR capture device except for the Disconnect control which is valid for all devices.

- Wrap Sequence Number:

  test what happens when you wrap the sequence number in struct v4l2_buffer around.

- Wrap Timestamp:

  test what happens when you wrap the timestamp in struct v4l2_buffer around.

- Percentage of Dropped Buffers:

  sets the percentage of buffers that are never returned by the driver (i.e., they are dropped).

- Disconnect:

emulates a USB disconnect. The device will act as if it has been disconnected. Only after all open filehandles to the device node have been closed will the device become 'connected' again.

- Inject V4L2_BUF_FLAG_ERROR:

  when pressed, the next frame returned by the driver will have the error flag set (i.e. the frame is marked corrupt).

- Inject VIDIOC_REQBUFS Error:

  when pressed, the next REQBUFS or CREATE_BUFS ioctl call will fail with an error. To be precise: the videobuf2 queue_setup() op will return -EINVAL.

- Inject VIDIOC_QBUF Error:

  when pressed, the next VIDIOC_QBUF or VIDIOC_PREPARE_BUFFER ioctl call will fail with an error. To be precise: the videobuf2 buf_prepare() op will return -EINVAL.

- Inject VIDIOC_STREAMON Error:

  when pressed, the next VIDIOC_STREAMON ioctl call will fail with an error. To be precise: the videobuf2 start_streaming() op will return -EINVAL.

- Inject Fatal Streaming Error:

  when pressed, the streaming core will be marked as having suffered a fatal error, the only way to recover from that is to stop streaming. To be precise: the videobuf2 vb2_queue_error() function is called.

VBI Raw Capture Controls

- Interlaced VBI Format:

  if set, then the raw VBI data will be interlaced instead of providing it grouped by field.

Digital Video Controls

- Rx RGB Quantization Range:

  sets the RGB quantization detection of the HDMI input. This combines with the Vivid 'Limited RGB Range (16-235)' control and can be used to test what happens if a source provides you with the wrong quantization range information. This can be tested by selecting an HDMI input, setting this control to Full or Limited range and selecting the opposite in the 'Limited RGB Range (16-235)' control. The effect is easy to see if the 'Gray Ramp' test pattern is selected.

- Tx RGB Quantization Range:

  sets the RGB quantization detection of the HDMI output. It is currently not used for anything in vivid, but most HDMI transmitters would typically have this control.

- Transmit Mode:

  sets the transmit mode of the HDMI output to HDMI or DVI-D. This affects the reported colorspace since DVI_D outputs will always use sRGB.

FM Radio Receiver Controls

- RDS Reception:

  set if the RDS receiver should be enabled.

- RDS Program Type:

- RDS PS Name:
- RDS Radio Text:
- RDS Traffic Announcement:
- RDS Traffic Program:
- RDS Music:

  these are all read-only controls. If RDS Rx I/O Mode is set to “Block I/O”, then they are inactive as well. If RDS Rx I/O Mode is set to “Controls”, then these controls report the received RDS data.

Note:

The vivid implementation of this is pretty basic: they are only updated when you set a new frequency or when you get the tuner status (VIDIOC_G_TUNER).

- Radio HW Seek Mode:

  can be one of “Bounded”, “Wrap Around” or “Both”. This determines if VIDIOC_S_HW_FREQ_SEEK will be bounded by the frequency range or wrap-around or if it is selectable by the user.

- Radio Programmable HW Seek:

  if set, then the user can provide the lower and upper bound of the HW Seek. Otherwise the frequency range boundaries will be used.

- Generate RBDS Instead of RDS:

  if set, then generate RBDS (the US variant of RDS) data instead of RDS (European-style RDS). This affects only the PICODE and PTY codes.

- RDS Rx I/O Mode:

  this can be “Block I/O” where the RDS blocks have to be read() by the application, or “Controls” where the RDS data is provided by the RDS controls mentioned above.

FM Radio Modulator Controls

- RDS Program ID:
- RDS Program Type:
- RDS PS Name:
- RDS Radio Text:
- RDS Stereo:
- RDS Artificial Head:
- RDS Compressed:
- RDS Dynamic PTY:
- RDS Traffic Announcement:
- RDS Traffic Program:
- RDS Music:

  these are all controls that set the RDS data that is transmitted by the FM modulator.

- RDS Tx I/O Mode:

this can be “Block I/O” where the application has to use write() to pass the RDS blocks to the driver, or “Controls” where the RDS data is Provided by the RDS controls mentioned above.

4.30.10 Video, VBI and RDS Looping

The vivid driver supports looping of video output to video input, VBI output to VBI input and RDS output to RDS input. For video/VBI looping this emulates as if a cable was hooked up between the output and input connector. So video and VBI looping is only supported between S-Video and HDMI inputs and outputs. VBI is only valid for S-Video as it makes no sense for HDMI.

Since radio is wireless this looping always happens if the radio receiver frequency is close to the radio transmitter frequency. In that case the radio transmitter will ‘override’ the emulated radio stations.

Looping is currently supported only between devices created by the same vivid driver instance.

Video and Sliced VBI looping

The way to enable video/VBI looping is currently fairly crude. A ‘Loop Video’ control is available in the “Vivid” control class of the video capture and VBI capture devices. When checked the video looping will be enabled. Once enabled any video S-Video or HDMI input will show a static test pattern until the video output has started. At that time the video output will be looped to the video input provided that:

- the input type matches the output type. So the HDMI input cannot receive video from the S-Video output.
- the video resolution of the video input must match that of the video output. So it is not possible to loop a 50 Hz (720x576) S-Video output to a 60 Hz (720x480) S-Video input, or a 720p60 HDMI output to a 1080p30 input.
- the pixel formats must be identical on both sides. Otherwise the driver would have to do pixel format conversion as well, and that’s taking things too far.
- the field settings must be identical on both sides. Same reason as above: requiring the driver to convert from one field format to another complicated matters too much. This also prohibits capturing with ‘Field Top’ or ‘Field Bottom’ when the output video is set to ‘Field Alternate’. This combination, while legal, became too complicated to support. Both sides have to be ‘Field Alternate’ for this to work. Also note that for this specific case the sequence and field counting in struct v4l2_buffer on the capture side may not be 100% accurate.
- field settings V4L2_FIELD_SEQ_TB/BT are not supported. While it is possible to implement this, it would mean a lot of work to get this right. Since these field values are rarely used the decision was made not to implement this for now.
- on the input side the “Standard Signal Mode” for the S-Video input or the “DV Timings Signal Mode” for the HDMI input should be configured so that a valid signal is passed to the video input.

The framerates do not have to match, although this might change in the future.

By default you will see the OSD text superimposed on top of the looped video. This can be turned off by changing the “OSD Text Mode” control of the video capture device.

For VBI looping to work all of the above must be valid and in addition the vbi output must be configured for sliced VBI. The VBI capture side can be configured for either raw or sliced VBI. Note that at the moment only CC/XDS (60 Hz formats) and WSS (50 Hz formats) VBI data is looped. Teletext VBI data is not looped.

Radio & RDS Looping

As mentioned in section 6 the radio receiver emulates stations are regular frequency intervals. Depending on the frequency of the radio receiver a signal strength value is calculated (this is returned by VIDIOC_G_TUNER). However, it will also look at the frequency set by the radio transmitter and if that results in a higher signal strength than the settings of the radio transmitter will be used as if it was a valid station.

This also includes the RDS data (if any) that the transmitter 'transmits'. This is received faithfully on the receiver side. Note that when the driver is loaded the frequencies of the radio receiver and transmitter are not identical, so initially no looping takes place.

4.30.11 Cropping, Composing, Scaling

This driver supports cropping, composing and scaling in any combination. Normally which features are supported can be selected through the Vivid controls, but it is also possible to hardcode it when the module is loaded through the ccs_cap_mode and ccs_out_mode module options. See section 1 on the details of these module options.

This allows you to test your application for all these variations.

Note that the webcam input never supports cropping, composing or scaling. That only applies to the TV/S-Video/HDMI inputs and outputs. The reason is that webcams, including this virtual implementation, normally use VIDIOC_ENUM_FRAMESIZES to list a set of discrete framesizes that it supports. And that does not combine with cropping, composing or scaling. This is primarily a limitation of the V4L2 API which is carefully reproduced here.

The minimum and maximum resolutions that the scaler can achieve are 16x16 and (4096 * 4) x (2160 x 4), but it can only scale up or down by a factor of 4 or less. So for a source resolution of 1280x720 the minimum the scaler can do is 320x180 and the maximum is 5120x2880. You can play around with this using the qv4l2 test tool and you will see these dependencies.

This driver also supports larger 'bytesperline' settings, something that VIDIOC_S_FMT allows but that few drivers implement.

The scaler is a simple scaler that uses the Coarse Bresenham algorithm. It's designed for speed and simplicity, not quality.

If the combination of crop, compose and scaling allows it, then it is possible to change crop and compose rectangles on the fly.

4.30.12 Formats

The driver supports all the regular packed and planar 4:4:4, 4:2:2 and 4:2:0 YUYV formats, 8, 16, 24 and 32 RGB packed formats and various multiplanar formats.

The alpha component can be set through the 'Alpha Component' User control for those formats that support it. If the 'Apply Alpha To Red Only' control is set, then the alpha component is only used for the color red and set to 0 otherwise.

The driver has to be configured to support the multiplanar formats. By default the driver instances are single-planar. This can be changed by setting the multiplanar module option, see section 1 for more details on that option.

If the driver instance is using the multiplanar formats/API, then the first single planar format (YUYV) and the multiplanar NV16M and NV61M formats the will have a plane that has a non-zero data_offset of 128 bytes. It is rare for data_offset to be non-zero, so this is a useful feature for testing applications.

Video output will also honor any data_offset that the application set.

4.30.13 Capture Overlay

Note: capture overlay support is implemented primarily to test the existing V4L2 capture overlay API. In practice few if any GPUs support such overlays anymore, and neither are they generally needed anymore since modern hardware is so much more capable. By setting flag 0x10000 in the node_types module option the vivid driver will create a simple framebuffer device that can be used for testing this API. Whether this API should be used for new drivers is questionable.

This driver has support for a destructive capture overlay with bitmap clipping and list clipping (up to 16 rectangles) capabilities. Overlays are not supported for multiplanar formats. It also honors the struct v4l2_window field setting: if it is set to FIELD_TOP or FIELD_BOTTOM and the capture setting is FIELD_ALTERNATE, then only the top or bottom fields will be copied to the overlay.

The overlay only works if you are also capturing at that same time. This is a vivid limitation since it copies from a buffer to the overlay instead of filling the overlay directly. And if you are not capturing, then no buffers are available to fill.

In addition, the pixelformat of the capture format and that of the framebuffer must be the same for the overlay to work. Otherwise VIDIOC_OVERLAY will return an error.

In order to really see what it going on you will need to create two vivid instances: the first with a framebuffer enabled. You configure the capture overlay of the second instance to use the framebuffer of the first, then you start capturing in the second instance. For the first instance you setup the output overlay for the video output, turn on video looping and capture to see the blended framebuffer overlay that's being written to by the second instance. This setup would require the following commands:

```
$ sudo modprobe vivid n_devs=2 node_types=0x10101,0x1
$ v4l2-ctl -d1 --find-fb
/dev/fb1 is the framebuffer associated with base address 0x12800000
$ sudo v4l2-ctl -d2 --set-fbuf fb=1
$ v4l2-ctl -d1 --set-fbuf fb=1
$ v4l2-ctl -d0 --set-fmt-video=pixelformat='AR15'
$ v4l2-ctl -d1 --set-fmt-video-out=pixelformat='AR15'
$ v4l2-ctl -d2 --set-fmt-video=pixelformat='AR15'
$ v4l2-ctl -d0 -i2
$ v4l2-ctl -d2 -i2
$ v4l2-ctl -d2 -c horizontal_movement=4
$ v4l2-ctl -d1 --overlay=1
$ v4l2-ctl -d1 -c loop_video=1
$ v4l2-ctl -d2 --stream-mmap --overlay=1
```

And from another console:

```
$ v4l2-ctl -d1 --stream-out-mmap
```

And yet another console:

```
$ qv4l2
```

and start streaming.

As you can see, this is not for the faint of heart...

4.30.14 Output Overlay

Note: output overlays are primarily implemented in order to test the existing V4L2 output overlay API. Whether this API should be used for new drivers is questionable.

This driver has support for an output overlay and is capable of:

- bitmap clipping,
- list clipping (up to 16 rectangles)
- chromakey
- source chromakey
- global alpha
- local alpha
- local inverse alpha

Output overlays are not supported for multiplanar formats. In addition, the pixelformat of the capture format and that of the framebuffer must be the same for the overlay to work. Otherwise VIDIOC_OVERLAY will return an error.

Output overlays only work if the driver has been configured to create a framebuffer by setting flag 0x10000 in the node_types module option. The created framebuffer has a size of 720x576 and supports ARGB 1:5:5:5 and RGB 5:6:5.

In order to see the effects of the various clipping, chromakeying or alpha processing capabilities you need to turn on video looping and see the results on the capture side. The use of the clipping, chromakeying or alpha processing capabilities will slow down the video loop considerably as a lot of checks have to be done per pixel.

4.30.15 CEC (Consumer Electronics Control)

If there are HDMI inputs then a CEC adapter will be created that has the same number of input ports. This is the equivalent of e.g. a TV that has that number of inputs. Each HDMI output will also create a CEC adapter that is hooked up to the corresponding input port, or (if there are more outputs than inputs) is not hooked up at all. In other words, this is the equivalent of hooking up each output device to an input port of the TV. Any remaining output devices remain unconnected.

The EDID that each output reads reports a unique CEC physical address that is based on the physical address of the EDID of the input. So if the EDID of the receiver has physical address A.B.0.0, then each output will see an EDID containing physical address A.B.C.0 where C is 1 to the number of inputs. If there are more outputs than inputs then the remaining outputs have a CEC adapter that is disabled and reports an invalid physical address.

4.30.16 Some Future Improvements

Just as a reminder and in no particular order:

- Add a virtual alsa driver to test audio
- Add virtual sub-devices and media controller support
- Some support for testing compressed video
- Add support to loop raw VBI output to raw VBI input
- Add support to loop teletext sliced VBI output to VBI input
- Fix sequence/field numbering when looping of video with alternate fields
- Add support for V4L2_CID_BG_COLOR for video outputs
- Add ARGB888 overlay support: better testing of the alpha channel
- Improve pixel aspect support in the tpg code by passing a real v4l2_fract
- Use per-queue locks and/or per-device locks to improve throughput
- Add support to loop from a specific output to a specific input across vivid instances
- The SDR radio should use the same ‘frequencies’ for stations as the normal radio receiver, and give back noise if the frequency doesn’t match up with a station frequency
- Make a thread for the RDS generation, that would help in particular for the “Controls” RDS Rx I/O Mode as the read-only RDS controls could be updated in real-time.
- Changing the EDID should cause hotplug detect emulation to happen.

4.31 The Zoran driver

unified zoran driver (zr360x7, zoran, buz, dc10(+), dc30(+), lml33)

website: http://mjpeg.sourceforge.net/driver-zoran/

4.31.1 Frequently Asked Questions

4.31.2 What cards are supported

Iomega Buz, Linux Media Labs LML33/LML33R10, Pinnacle/Miro DC10/DC10+/DC30/DC30+ and related boards (available under various names).

Iomega Buz

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Philips saa7111 TV decoder
- Philips saa7185 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, saa7111, saa7185, zr36060, zr36067

Inputs/outputs: Composite and S-video

Norms: PAL, SECAM (720x576 @ 25 fps), NTSC (720x480 @ 29.97 fps)

Card number: 7

AverMedia 6 Eyes AVS6EYES

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Samsung ks0127 TV decoder
- Conexant bt866 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, ks0127, bt866, zr36060, zr36067

Inputs/outputs: Six physical inputs. 1-6 are composite, 1-2, 3-4, 5-6 doubles as S-video, 1-3 triples as component. One composite output.

Norms: PAL, SECAM (720x576 @ 25 fps), NTSC (720x480 @ 29.97 fps)

Card number: 8

Note:

Not autodetected, card=8 is necessary.

Linux Media Labs LML33

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Brooktree bt819 TV decoder

- Brooktree bt856 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, bt819, bt856, zr36060, zr36067
Inputs/outputs: Composite and S-video
Norms: PAL (720x576 @ 25 fps), NTSC (720x480 @ 29.97 fps)
Card number: 5

Linux Media Labs LML33R10

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Philips saa7114 TV decoder
- Analog Devices adv7170 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, saa7114, adv7170, zr36060, zr36067
Inputs/outputs: Composite and S-video
Norms: PAL (720x576 @ 25 fps), NTSC (720x480 @ 29.97 fps)
Card number: 6

Pinnacle/Miro DC10(new)

- Zoran zr36057 PCI controller
- Zoran zr36060 MJPEG codec
- Philips saa7110a TV decoder
- Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, saa7110, adv7175, zr36060, zr36067
Inputs/outputs: Composite, S-video and Internal
Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)
Card number: 1

Pinnacle/Miro DC10+

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Philips saa7110a TV decoder
- Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, sa7110, adv7175, zr36060, zr36067
Inputs/outputs: Composite, S-video and Internal
Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)
Card number: 2

Pinnacle/Miro DC10(old)

- Zoran zr36057 PCI controller
- Zoran zr36050 MJPEG codec
- Zoran zr36016 Video Front End or Fuji md0211 Video Front End (clone?)
- Micronas vpx3220a TV decoder
- mse3000 TV encoder or Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, vpx3220, mse3000/adv7175, zr36050, zr36016, zr36067

Inputs/outputs: Composite, S-video and Internal

Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)

Card number: 0

Pinnacle/Miro DC30

- Zoran zr36057 PCI controller
- Zoran zr36050 MJPEG codec
- Zoran zr36016 Video Front End
- Micronas vpx3225d/vpx3220a/vpx3216b TV decoder
- Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, vpx3220/vpx3224, adv7175, zr36050, zr36016, zr36067

Inputs/outputs: Composite, S-video and Internal

Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)

Card number: 3

Pinnacle/Miro DC30+

- Zoran zr36067 PCI controller
- Zoran zr36050 MJPEG codec
- Zoran zr36016 Video Front End
- Micronas vpx3225d/vpx3220a/vpx3216b TV decoder
- Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, vpx3220/vpx3224, adv7175, zr36050, zr36015, zr36067

Inputs/outputs: Composite, S-video and Internal

Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)

Card number: 4

Note:

1. *No module for the mse3000 is available yet*
2. *No module for the vpx3224 is available yet*

4.31.3 1.1 What the TV decoder can do an what not

The best know TV standards are NTSC/PAL/SECAM. but for decoding a frame that information is not enough. There are several formats of the TV standards. And not every TV decoder is able to handle every format. Also the every combination is supported by the driver. There are currently 11 different tv broadcast formats all aver the world.

The CCIR defines parameters needed for broadcasting the signal. The CCIR has defined different standards: A,B,D,E,F,G,D,H,I,K,K1,L,M,N,... The CCIR says not much about the colorsystem used !!! And talking about a colorsystem says not to much about how it is broadcast.

The CCIR standards A,E,F are not used any more.

When you speak about NTSC, you usually mean the standard: CCIR - M using the NTSC colorsystem which is used in the USA, Japan, Mexico, Canada and a few others.

When you talk about PAL, you usually mean: CCIR - B/G using the PAL colorsystem which is used in many Countries.

When you talk about SECAM, you mean: CCIR - L using the SECAM Colorsystem which is used in France, and a few others.

There the other version of SECAM, CCIR - D/K is used in Bulgaria, China, Slovakai, Hungary, Korea (Rep.), Poland, Rumania and a others.

The CCIR - H uses the PAL colorsystem (sometimes SECAM) and is used in Egypt, Libya, Sri Lanka, Syrain Arab. Rep.

The CCIR - I uses the PAL colorsystem, and is used in Great Britain, Hong Kong, Ireland, Nigeria, South Africa.

The CCIR - N uses the PAL colorsystem and PAL frame size but the NTSC framerate, and is used in Argentinia, Uruguay, an a few others

We do not talk about how the audio is broadcast !

A rather good sites about the TV standards are: http://www.sony.jp/support/ http://info.electronicwerkstatt.de/bereiche/fernsehtechnik/frequenzen_und_normen/Fernsehnormen/ and http://www.cabl.com/restaurant/channel.html

Other weird things around: NTSC 4.43 is a modificated NTSC, which is mainly used in PAL VCR's that are able to play back NTSC. PAL 60 seems to be the same as NTSC 4.43 . The Datasheets also talk about NTSC 44, It seems as if it would be the same as NTSC 4.43. NTSC Combs seems to be a decoder mode where the decoder uses a comb filter to split coma and luma instead of a Delay line.

But I did not defiantly find out what NTSC Comb is.

Philips saa7111 TV decoder

- was introduced in 1997, is used in the BUZ and
- can handle: PAL B/G/H/I, PAL N, PAL M, NTSC M, NTSC N, NTSC 4.43 and SECAM

Philips saa7110a TV decoder

- was introduced in 1995, is used in the Pinnacle/Miro DC10(new), DC10+ and
- can handle: PAL B/G, NTSC M and SECAM

Philips saa7114 TV decoder

- was introduced in 2000, is used in the LML33R10 and
- can handle: PAL B/G/D/H/I/N, PAL N, PAL M, NTSC M, NTSC 4.43 and SECAM

Brooktree bt819 TV decoder

- was introduced in 1996, and is used in the LML33 and
- can handle: PAL B/D/G/H/I, NTSC M

Micronas vpx3220a TV decoder

- was introduced in 1996, is used in the DC30 and DC30+ and
- can handle: PAL B/G/H/I, PAL N, PAL M, NTSC M, NTSC 44, PAL 60, SECAM,NTSC Comb

Samsung ks0127 TV decoder

- is used in the AVS6EYES card and
- can handle: NTSC-M/N/44, PAL-M/N/B/G/H/I/D/K/L and SECAM

4.31.4 What the TV encoder can do an what not

The TV encoder are doing the “same” as the decoder, but in the oder direction. You feed them digital data and the generate a Composite or SVHS signal. For information about the colorsystems and TV norm take a look in the TV decoder section.

Philips saa7185 TV Encoder

- was introduced in 1996, is used in the BUZ
- can generate: PAL B/G, NTSC M

Brooktree bt856 TV Encoder

- was introduced in 1994, is used in the LML33
- can generate: PAL B/D/G/H/I/N, PAL M, NTSC M, PAL-N (Argentina)

Analog Devices adv7170 TV Encoder

- was introduced in 2000, is used in the LML300R10
- can generate: PAL B/D/G/H/I/N, PAL M, NTSC M, PAL 60

Analog Devices adv7175 TV Encoder

- was introduced in 1996, is used in the DC10, DC10+, DC10 old, DC30, DC30+
- can generate: PAL B/D/G/H/I/N, PAL M, NTSC M

ITT mse3000 TV encoder

- was introduced in 1991, is used in the DC10 old
- can generate: PAL , NTSC , SECAM

Conexant bt866 TV encoder

- is used in AVS6EYES, and
- can generate: NTSC/PAL, PALM, PALN

The adv717x, should be able to produce PAL N. But you find nothing PAL N specific in the registers. Seem that you have to reuse a other standard to generate PAL N, maybe it would work if you use the PAL M settings.

4.31.5 How do I get this damn thing to work

Load zr36067.o. If it can't autodetect your card, use the card=X insmod option with X being the card number as given in the previous section. To have more than one card, use card=X1[,X2[,X3,[X4[..]]]]

To automate this, add the following to your /etc/modprobe.d/zoran.conf:

options zr36067 card=X1[,X2[,X3[,X4[..]]]] alias char-major-81-0 zr36067

One thing to keep in mind is that this doesn't load zr36067.o itself yet. It just automates loading. If you start using xawtv, the device won't load on some systems, since you're trying to load modules as a user, which is not allowed ("permission denied"). A quick workaround is to add 'Load "v4l"' to XF86Config-4 when you use X by default, or to run 'v4l-conf -c <device>' in one of your startup scripts (normally rc.local) if you don't use X. Both make sure that the modules are loaded on startup, under the root account.

4.31.6 What mainboard should I use (or why doesn't my card work)

<insert lousy disclaimer here>. In short: good=SiS/Intel, bad=VIA.

Experience tells us that people with a Buz, on average, have more problems than users with a DC10+/LML33. Also, it tells us that people owning a VIA- based mainboard (ktXXX, MVP3) have more problems than users with a mainboard based on a different chipset. Here's some notes from Andrew Stevens:

Here's my experience of using LML33 and Buz on various motherboards:

- **VIA MVP3**
  - Forget it. Pointless. Doesn't work.
- **Intel 430FX (Pentium 200)**
  - LML33 perfect, Buz tolerable (3 or 4 frames dropped per movie)
- **Intel 440BX (early stepping)**
  - LML33 tolerable. Buz starting to get annoying (6-10 frames/hour)
- **Intel 440BX (late stepping)**
  - Buz tolerable, LML3 almost perfect (occasional single frame drops)
- **SiS735**
  - LML33 perfect, Buz tolerable.
- **VIA KT133(*)**
  - LML33 starting to get annoying, Buz poor enough that I have up.
- Both 440BX boards were dual CPU versions.

Bernhard Praschinger later added:

- **AMD 751**
  - Buz perfect-tolerable

- **AMD 760**
  - Buz perfect-tolerable

In general, people on the user mailinglist won't give you much of a chance if you have a VIA-based motherboard. They may be cheap, but sometimes, you'd rather want to spend some more money on better boards. In general, VIA mainboard's IDE/PCI performance will also suck badly compared to others. You'll noticed the DC10+/DC30+ aren't mentioned anywhere in the overview. Basically, you can assume that if the Buz works, the LML33 will work too. If the LML33 works, the DC10+/DC30+ will work too. They're most tolerant to different mainboard chipsets from all of the supported cards.

If you experience timeouts during capture, buy a better mainboard or lower the quality/buffersize during capture (see 'Concerning buffer sizes, quality, output size etc.'). If it hangs, there's little we can do as of now. Check your IRQs and make sure the card has its own interrupts.

4.31.7 Programming interface

This driver conforms to video4linux2. Support for V4L1 and for the custom zoran ioctls has been removed in kernel 2.6.38.

For programming example, please, look at lavrec.c and lavplay.c code in the MJPEG-tools (http://mjpeg.sf.net/).

Additional notes for software developers:

> The driver returns maxwidth and maxheight parameters according to the current TV standard (norm). Therefore, the software which communicates with the driver and "asks" for these parameters should first set the correct norm. Well, it seems logically correct: TV standard is "more constant" for current country than geometry settings of a variety of TV capture cards which may work in ITU or square pixel format.

4.31.8 Applications

Applications known to work with this driver:

TV viewing:

- xawtv
- kwintv
- probably any TV application that supports video4linux or video4linux2.

MJPEG capture/playback:

- mjpegtools/lavtools (or Linux Video Studio)
- gstreamer
- mplayer

General raw capture:

- xawtv
- gstreamer
- probably any application that supports video4linux or video4linux2

Video editing:

- Cinelerra
- MainActor
- mjpegtools (or Linux Video Studio)

4.31.9 Concerning buffer sizes, quality, output size etc.

The zr36060 can do 1:2 JPEG compression. This is really the theoretical maximum that the chipset can reach. The driver can, however, limit compression to a maximum (size) of 1:4. The reason for this is that some cards (e.g. Buz) can't handle 1:2 compression without stopping capture after only a few minutes. With 1:4, it'll mostly work. If you have a Buz, use 'low_bitrate=1' to go into 1:4 max. compression mode.

100% JPEG quality is thus 1:2 compression in practice. So for a full PAL frame (size 720x576). The JPEG fields are stored in YUY2 format, so the size of the fields are 720x288x16/2 bits/field (2 fields/frame) = 207360 bytes/field x 2 = 414720 bytes/frame (add some more bytes for headers and DHT (huffman)/DQT (quantization) tables, and you'll get to something like 512kB per frame for 1:2 compression. For 1:4 compression, you'd have frames of half this size.

Some additional explanation by Martin Samuelsson, which also explains the importance of buffer sizes: –> Hmm, I do not think it is really that way. With the current (downloaded > at 18:00 Monday) driver I get that output sizes for 10 sec: > -q 50 -b 128 : 24.283.332 Bytes > -q 50 -b 256 : 48.442.368 > -q 25 -b 128 : 24.655.992 > -q 25 -b 256 : 25.859.820

I woke up, and can't go to sleep again. I'll kill some time explaining why this doesn't look strange to me.

Let's do some math using a width of 704 pixels. I'm not sure whether the Buz actually use that number or not, but that's not too important right now.

704x288 pixels, one field, is 202752 pixels. Divided by 64 pixels per block; 3168 blocks per field. Each pixel consist of two bytes; 128 bytes per block; 1024 bits per block. 100% in the new driver mean 1:2 compression; the maximum output becomes 512 bits per block. Actually 510, but 512 is simpler to use for calculations.

Let's say that we specify d1q50. We thus want 256 bits per block; times 3168 becomes 811008 bits; 101376 bytes per field. We're talking raw bits and bytes here, so we don't need to do any fancy corrections for bits-per-pixel or such things. 101376 bytes per field.

d1 video contains two fields per frame. Those sum up to 202752 bytes per frame, and one of those frames goes into each buffer.

But wait a second! -b128 gives 128kB buffers! It's not possible to cram 202752 bytes of JPEG data into 128kB!

This is what the driver notice and automatically compensate for in your examples. Let's do some math using this information:

128kB is 131072 bytes. In this buffer, we want to store two fields, which leaves 65536 bytes for each field. Using 3168 blocks per field, we get 20.68686868... available bytes per block; 165 bits. We can't allow the request for 256 bits per block when there's only 165 bits available! The -q50 option is silently overridden, and the -b128 option takes precedence, leaving us with the equivalence of -q32.

This gives us a data rate of 165 bits per block, which, times 3168, sums up to 65340 bytes per field, out of the allowed 65536. The current driver has another level of rate limiting; it won't accept -q values that fill more than 6/8 of the specified buffers. (I'm not sure why. "Playing it safe" seem to be a safe bet. Personally, I think I would have lowered requested-bits-per-block by one, or something like that.) We can't use 165 bits per block, but have to lower it again, to 6/8 of the available buffer space: We end up with 124 bits per block, the equivalence of -q24. With 128kB buffers, you can't use greater than -q24 at -d1. (And PAL, and 704 pixels width...)

The third example is limited to -q24 through the same process. The second example, using very similar calculations, is limited to -q48. The only example that actually grab at the specified -q value is the last one, which is clearly visible, looking at the file size. –

Conclusion: the quality of the resulting movie depends on buffer size, quality, whether or not you use 'low_bitrate=1' as insmod option for the zr36060.c module to do 1:4 instead of 1:2 compression, etc.

If you experience timeouts, lowering the quality/buffersize or using 'low_bitrate=1 as insmod option for zr36060.o might actually help, as is proven by the Buz.

4.31.10 It hangs/crashes/fails/whatevers! Help!

Make sure that the card has its own interrupts (see /proc/interrupts), check the output of dmesg at high verbosity (load zr36067.o with debug=2, load all other modules with debug=1). Check that your mainboard is favorable (see question 2) and if not, test the card in another computer. Also see the notes given in question 3 and try lowering quality/buffersize/capturesize if recording fails after a period of time.

If all this doesn't help, give a clear description of the problem including detailed hardware information (memory+brand, mainboard+chipset+brand, which MJPEG card, processor, other PCI cards that might be of interest), give the system PnP information (/proc/interrupts, /proc/dma, /proc/devices), and give the kernel version, driver version, glibc version, gcc version and any other information that might possibly be of interest. Also provide the dmesg output at high verbosity. See 'Contacting' on how to contact the developers.

4.31.11 Maintainers/Contacting

The driver is currently maintained by Laurent Pinchart and Ronald Bultje (<laurent.pinchart@skynet.be> and <rbultje@ronald.bitfreak.net>). For bug reports or questions, please contact the mailinglist instead of the developers individually. For user questions (i.e. bug reports or how-to questions), send an email to <mjpeg-users@lists.sf.net>, for developers (i.e. if you want to help programming), send an email to <mjpeg-developer@lists.sf.net>. See http://www.sf.net/projects/mjpeg/ for subscription information.

For bug reports, be sure to include all the information as described in the section 'It hangs/crashes/fails/whatevers! Help!'. Please make sure you're using the latest version (http://mjpeg.sf.net/driver-zoran/).

Previous maintainers/developers of this driver include Serguei Miridonov <mirsev@cicese.mx>, Wolfgang Scherr <scherr@net4you.net>, Dave Perks <dperks@ibm.net> and Rainer Johanni <Rainer@Johanni.de>.

4.31.12 Driver's License

> This driver is distributed under the terms of the General Public License.
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

See http://www.gnu.org/ for more information.

4.32 Zoran 364xx based USB webcam module

site: http://royale.zerezo.com/zr364xx/

mail: royale@zerezo.com

Note:

This documentation is outdated

4.32.1 Introduction

This brings support under Linux for the Aiptek PocketDV 3300 in webcam mode. If you just want to get on your PC the pictures and movies on the camera, you should use the usb-storage module instead.

The driver works with several other cameras in webcam mode (see the list below).

Maybe this code can work for other JPEG/USB cams based on the Coach chips from Zoran?

Possible chipsets are : ZR36430 (ZR36430BGC) and maybe ZR36431, ZR36440, ZR36442...

You can try the experience changing the vendor/product ID values (look at the source code).

You can get these values by looking at /var/log/messages when you plug your camera, or by typing : cat /proc/bus/usb/devices.

If you manage to use your cam with this code, you can send me a mail (royale@zerezo.com) with the name of your cam and a patch if needed.

This is a beta release of the driver. Since version 0.70, this driver is only compatible with V4L2 API and 2.6.x kernels. If you need V4L1 or 2.4x kernels support, please use an older version, but the code is not maintained anymore. Good luck!

4.32.2 Install

In order to use this driver, you must compile it with your kernel.

Location: Device Drivers -> Multimedia devices -> Video For Linux -> Video Capture Adapters -> V4L USB devices

4.32.3 Usage

modprobe zr364xx debug=X mode=Y

- debug : set to 1 to enable verbose debug messages
- mode : 0 = 320x240, 1 = 160x120, 2 = 640x480

You can then use the camera with V4L2 compatible applications, for example Ekiga.

To capture a single image, try this: dd if=/dev/video0 of=test.jpg bs=1M count=1

4.32.4 links

http://mxhaard.free.fr/ (support for many others cams including some Aiptek PocketDV) http://www.harmwal.nl/pccam880/ (this project also supports cameras based on this chipset)

4.32.5 Supported devices

Vendor	Product	Distributor	Model
0x08ca	0x0109	Aiptek	PocketDV 3300
0x08ca	0x0109	Maxell	Maxcam PRO DV3
0x041e	0x4024	Creative	PC-CAM 880
0x0d64	0x0108	Aiptek	Fidelity 3200
0x0d64	0x0108	Praktica	DCZ 1.3 S
0x0d64	0x0108	Genius	Digital Camera (?)
0x0d64	0x0108	DXG Technology	Fashion Cam
0x0546	0x3187	Polaroid	iON 230

Continued on next page

Table 4.10 – continued from previous page

Vendor	Product	Distributor	Model
0x0d64	0x3108	Praktica	Exakta DC 2200
0x0d64	0x3108	Genius	G-Shot D211
0x0595	0x4343	Concord	Eye-Q Duo 1300
0x0595	0x4343	Concord	Eye-Q Duo 2000
0x0595	0x4343	Fujifilm	EX-10
0x0595	0x4343	Ricoh	RDC-6000
0x0595	0x4343	Digitrex	DSC 1300
0x0595	0x4343	Firstline	FDC 2000
0x0bb0	0x500d	Concord	EyeQ Go Wireless
0x0feb	0x2004	CRS Electronic	3.3 Digital Camera
0x0feb	0x2004	Packard Bell	DSC-300
0x055f	0xb500	Mustek	MDC 3000
0x08ca	0x2062	Aiptek	PocketDV 5700
0x052b	0x1a18	Chiphead	Megapix V12
0x04c8	0x0729	Konica	Revio 2
0x04f2	0xa208	Creative	PC-CAM 850
0x0784	0x0040	Traveler	Slimline X5
0x06d6	0x0034	Trust	Powerc@m 750
0x0a17	0x0062	Pentax	Optio 50L
0x06d6	0x003b	Trust	Powerc@m 970Z
0x0a17	0x004e	Pentax	Optio 50
0x041e	0x405d	Creative	DiVi CAM 516
0x08ca	0x2102	Aiptek	DV T300
0x06d6	0x003d	Trust	Powerc@m 910Z

CHAPTER

FIVE

INDICES

- genindex