

Video4Linux (V4L) Programming Manual

V4L developers(Alan Cox), Linux Media Labs

29th August 2001

Contents

1 Introduction	1
2 Devices	1
3 Capability Query Ioctl	1
4 Frame Buffer	2
5 Capture Windows	3
6 Video Sources	3
7 Image Properties	4
8 Tuning	5
9 Audio	6
10 Reading Images	6
10.1 Reading Images using 'read' syscall.	6
10.2 Reading Images using mmap interface.	6
11 Miscellaneous	7

1 Introduction

This document is based on <http://roadrunner.swansea.uk.linux.org/v4l.shtml>. We (Linux Media Labs) cleaned it up and added some clarifications. Feel free to contact us at v4l@linuxmedialabs.com. Source for this document can be obtained at:

```
cvs://pserver:vleo@cvs.otlinux.ru:/home/cvs/public  
(login password: public, checkout LMLCD/LinTV/doc)
```

2 Devices

Video4Linux provides the following sets of device files. These live on the character device formerly known as `"/dev/bttv"`. `/dev/bttv` should be a symlink to `/dev/video0` for most people:

Device Name	Minor Range	Function
<code>/dev/video</code>	0-63	Video Capture Interface
<code>/dev/radio</code>	64-127	AM/FM Radio Devices
<code>/dev/vtx</code>	192-233	Teletext Interface Chips
<code>/dev/vbi</code>	224-239	Raw VBI Data (Intercast/teletext)

Video4Linux programs open and scan the devices to find what they are looking for. Capability queries define what each interface supports. The described API is only defined for video capture cards. The relevant subset applies to radio cards. Teletext interfaces talk the existing VTX API.

3 Capability Query Ioctl

The **VIDIOCGCAP** ioctl call is used to obtain the capability information for a video device. The **struct video_capability** object passed to the ioctl is completed and returned. It contains the following information:

Field	Description
<code>char name[32]</code>	Canonical name for this interface
<code>int type</code>	Type of interface
<code>int channels</code>	Number of radio/tv channels if appropriate
<code>int audios</code>	Number of audio devices if appropriate
<code>int maxwidth</code>	Maximum capture width in pixels
<code>int maxheight</code>	Maximum capture height in pixels
<code>int minwidth</code>	Minimum capture width in pixels
<code>int minheight</code>	Minimum capture height in pixels

The type field lists the capability flags for the device. These are as follows:

Name	Description
<code>VID_TYPE_CAPTURE</code>	Can capture to memory
<code>VID_TYPE_TUNER</code>	Has a tuner of some form
<code>VID_TYPE_TELETEXT</code>	Has teletext capability
<code>VID_TYPE_OVERLAY</code>	Can overlay its image onto the frame buffer
<code>VID_TYPE_CHROMAKEY</code>	Overlay is Chromakeyed
<code>VID_TYPE_CLIPPING</code>	Overlay clipping is supported
<code>VID_TYPE_FRAMERAM</code>	Overlay overwrites frame buffer memory
<code>VID_TYPE_SCALES</code>	The hardware supports image scaling
<code>VID_TYPE_MONOCHROME</code>	Image capture is grey scale only
<code>VID_TYPE_SUBCAPTURE</code>	Capture can be of only part of the image

The minimum and maximum sizes listed for a capture device do not imply all that all height/width ratios or sizes within the range are possible. A request to set a size will be honoured by the largest available capture size whose capture is no large than the requested rectangle in either direction. For example the quickcam has 3 fixed settings.

4 Frame Buffer

Capture cards that drop data directly onto the frame buffer must be told the base address of the frame buffer, its size and organisation. This is a privileged ioctl and one that eventually X itself should set.

The **VIDIOCSFBUF** ioctl sets the frame buffer parameters for a capture card. If the card does not do direct writes to the frame buffer then this ioctl will be unsupported. The **VIDIOCGFBUF** ioctl returns the currently used parameters. The structure used in both cases is a **struct video_buffer**:

Field	Description
void *base	Base physical address of the buffer
int height	Height of the frame buffer
int width	Width of the frame buffer
int depth	Depth of the frame buffer
int bytesperline	Number of bytes of memory between the start of two adjacent lines

Note that these values reflect the physical layout of the frame buffer. The visible area may be smaller. In fact under XFree86 this is commonly the case. XFree86 DGA can provide the parameters required to set up this ioctl. Setting the base address to NULL indicates there is no physical frame buffer access.

5 Capture Windows

The capture area is described by a **struct video_window**. This defines a capture area and the clipping information, if relevant. The **VIDIOCGWIN** ioctl recovers the current settings and the **VIDIOCSWIN** sets new values. A successful call to **VIDIOCSWIN** indicates that a suitable set of parameters have been chosen. They do not indicate that exactly what was requested was granted. The program should call **VIDIOCGWIN** to check if the nearest match was suitable. The **struct video_window** contains the following fields:

__u32 x	The X co-ordinate specified in X windows format.
__u32 y	The Y co-ordinate specified in X windows format.
__u32 width	The width of the image capture.
__u32 height	The height of the image capture.
__u32 chromakey	A host order RGB32 value for the chroma key.
__u32 flags	Additional capture flags.
struct video_clip *clips	A list of clipping rectangles. (Set only)
int clipcount	The number of clipping rectangles. (Set only)

Clipping rectangles are passed as an array. Each **struct video_clip** consists of the following fields available to the user:

__s32 x	X co-ordinate of rectangle to skip
__s32 y	Y co-ordinate of rectangle to skip
__s32 width	Width of rectangle to skip
__s32 height	Height of rectangle to skip
struct video_clip*	Pointer to next clip in list

Merely setting the window does not enable capturing. Overlay capturing is activated by passing the **VIDIOCCAPTURE** ioctl a value of 1, and disabled by passing it a value of 0.

Some capture devices can capture a subfield of the image they actually see. This is indicated when **VIDEO_TYPE_SUBCAPTURE** is defined. The **video_capture** describes the time and spacial subfields to capture. The **video_capture** structure contains the following fields:

__u32 x	X co-ordinate of source rectangle to grab
__u32 y	Y co-ordinate of source rectangle to grab
__u32 width	Width of source rectangle to grab
__u32 height	Height of source rectangle to grab
__u16 decimation	Decimation to apply
__u16 flags	Flag settings for grabbing

The available flags are:

Name	Description
VIDEO_CAPTURE_ODD	Capture only odd frames
VIDEO_CAPTURE_EVEN	Capture only even frames

6 Video Sources

Each video4linux video or audio device captures from one or more source channels. Each channel can be queried with the **VIDIOCGCHAN** ioctl call. Before invoking this function the caller must set the channel field to the channel that is being queried. On return the **struct video_channel** is filled in with information about the nature of the channel itself.

The **VIDIOCSCHAN** ioctl takes a **struct video_channel** argument and switches the capture parameters (channel number and video signal norm) to the values specified in **struct video_channel** fields. It is not defined whether parameters such as colour settings or tuning are maintained across a channel switch. The caller should maintain settings as desired for each channel. (This is reasonable as different video inputs may have different properties).

The **struct video_channel** consists of the following:

Field	Description
int channel	The channel number
char name[32]	The input name - preferably reflecting the label on the card input itself
int tuners	Number of tuners for this input
__u32 flags	Properties the tuner has
__u32 type	Input type (if known)
__u32 norm	The norm for this channel

The flags defined are:

VIDEO_VC_TUNER	Channel has tuners.
VIDEO_VC_AUDIO	Channel has audio.
VIDEO_VC_NORM	Channel has norm setting.

The types defined are:

VIDEO_TYPE_TV	The input is a TV input.
VIDEO_TYPE_CAMERA	The input is a camera.

The norms defined are:

VIDEO_MODE_PAL	The video is in PAL mode
VIDEO_MODE_NTSC	The video is in NTSC mode
VIDEO_MODE_SECAM	The video is in SECAM mode
VIDEO_MODE_AUTO	The video mode auto switches, or mode does not apply

7 Image Properties

The image properties of the picture can be queried with the **VIDIOCGPICT** ioctl which fills in a **struct video_picture**. The **VIDIOCSPICT** ioctl allows values to be changed. All values except for the palette type are scaled between 0-65535.

The **struct video_picture** consists of the following fields:

Field	Description
<code>__u16 brightness</code>	Picture brightness
<code>__u16 hue</code>	Picture hue (colour only)
<code>__u16 colour</code>	Picture colour (colour only)
<code>__u16 contrast</code>	Picture contrast
<code>__u16 whiteness</code>	The whiteness (greyscale only)
<code>__u16 depth</code>	The capture depth (may need to match the frame buffer depth)
<code>__u16 palette</code>	Reports the palette that should be used for this image

The following palettes are defined:

<code>VIDEO_PALETTE_GREY</code>	Linear intensity grey scale (255 is brightest).
<code>VIDEO_PALETTE_HI240</code>	The BT848 8bit colour cube.
<code>VIDEO_PALETTE_RGB565</code>	RGB565 packed into 16 bit words.
<code>VIDEO_PALETTE_RGB555</code>	RGV555 packed into 16 bit words, top bit undefined.
<code>VIDEO_PALETTE_RGB24</code>	RGB888 packed into 24bit words.
<code>VIDEO_PALETTE_RGB32</code>	RGB888 packed into the low 3 bytes of 32bit words.
	The top 8bits are undefined.
<code>VIDEO_PALETTE_YUV422</code>	Video style YUV422 - 8bits packed 4bits Y 2bits U 2bits V
<code>VIDEO_PALETTE_YUYV</code>	Describe me
<code>VIDEO_PALETTE_UYVY</code>	Describe me
<code>VIDEO_PALETTE_YUV420</code>	YUV420 capture
<code>VIDEO_PALETTE_YUV411</code>	YUV411 capture
<code>VIDEO_PALETTE_RAW</code>	RAW capture (BT848)
<code>VIDEO_PALETTE_YUV411PYUV</code>	4:1:1 Planar
<code>VIDEO_PALETTE_YUV422PYUV</code>	4:2:2 Planar

8 Tuning

Each video input channel can have one or more tuners associated with it. Many devices will not have tuners. TV cards and radio cards will have one or more tuners attached.

Tuners are described by a **struct video_tuner** which can be obtained by the **VIDIOCG-TUNER** ioctl. Fill in the tuner number in the structure then pass the structure to the ioctl to have the data filled in. The tuner can be switched using **VIDIOCSTUNER** which takes an integer argument giving the tuner to use.

A **struct video_tuner** has the following fields:

<code>int tuner</code>	Number of the tuner
<code>char name[32]</code>	Cannonical name for this tuner (eg FM/AM/TV)
<code>ulong rangelow</code>	Lowest tunable frequency
<code>ulong rangehigh</code>	Highest tunable frequency
<code>__u32 flags</code>	Flags describing the tuner
<code>__u16 mode</code>	The video signal mode if relevant
<code>__u16 signal</code>	Signal strength if known - between 0-65535

The following flags exist:

<code>VIDEO_TUNER_PAL</code>	PAL tuning is supported
<code>VIDEO_TUNER_NTSC</code>	NTSC tuning is supported
<code>VIDEO_TUNER_SECAM</code>	SECAM tuning is supported
<code>VIDEO_TUNER_LOW</code>	Frequency is in a lower range
<code>VIDEO_TUNER_NORM</code>	The norm for this tuner is settable
<code>VIDEO_TUNER_STEREO_ON</code>	The tuner is seeing stereo audio

The following modes are defined:

VIDEO_MODE_PAL	The tuner is in PAL mode
VIDEO_MODE_NTSC	The tuner is in NTSC mode
VIDEO_MODE_SECAM	The tuner is in SECAM mode
VIDEO_MODE_AUTO	The tuner auto switches, or mode does not apply

Tuning frequencies are an unsigned 32bit value in 1/16th MHz or if the **VIDEO_TUNER_LOW** flag is set they are in 1/16th KHz. The current frequency is obtained as an unsigned long via the **VIDIOCGFREQ** ioctl and set by the **VIDIOCSFREQ** ioctl.

9 Audio

TV and Radio devices have one or more audio inputs that may be selected. The audio properties are queried by passing a **struct video_audio** to **VIDIOCGAUDIO** ioctl. The **VIDIOCSAUDIO** ioctl sets audio properties.

The structure contains the following fields:

__u16 audio	The channel number
__u16 volume	The volume level
__u16 bass	The bass level
__u16 treble	The treble level
__u32 flags	Flags describing the audio channel
char name[16]	Canonical name for the audio input
__u16 mode	The mode the audio input is in
__u16 balance	The left/right balance
__u16 step	Actual step used by the hardware

The following flags are defined:

VIDEO_AUDIO_MUTE	The audio is muted
VIDEO_AUDIO_MUTABLE	Audio muting is supported
VIDEO_AUDIO_VOLUME	The volume is controllable
VIDEO_AUDIO_BASS	The bass is controllable
VIDEO_AUDIO_TREBLE	The treble is controllable
VIDEO_AUDIO_BALANCE	The balance is controllable

The following decoding modes are defined

VIDEO_SOUND_MONO	Mono signal
VIDEO_SOUND_STEREO	Stereo signal (NICAM for TV)
VIDEO_SOUND_LANG1	European TV alternate language 1
VIDEO_SOUND_LANG2	European TV alternate language 2

10 Reading Images

There are two ways of acquiring images from the grab device. First is using the **read** syscall, second is through **mmap** interface.

10.1 Reading Images using 'read' syscall.

Each call to the **read** syscall returns the next available image from the device. It is up to the caller to set the format and then to pass a suitable size buffer and length to the function. Not all devices will support read operations.

10.2 Reading Images using mmap interface.

(Terminology issue - the “frame” term used here refers to what is called a “buffer” in the double-buffering systems, the “buffer” term means the whole collection of “frames“.)

To use the **mmap** interface a user must set up the buffer first. This is done by issuing **VIDIOCGMBUF** ioctl to tell the driver the size of buffer to mmap, the number of frames to use and the offset within the buffer for each frame. These properties are passed through the **video_mbuf** structure, shown below. The number of frames supported is device dependant and may only be one.

The **video_mbuf** structure contains the following fields:

int size	The number of bytes to map.
int frames	The number of frames.
int offsets[]	The offset of each frame.

Next, mmap should be called to map device onto memory. Once the mmap has been made, the frames management is performed through two ioctls: **VIDIOCMCAPTURE** and **VIDIOCSYNC**. **VIDIOCMCAPTURE** ioctl carries two functions for some reason - sets the image size you wish to use (which should match or be below the initial query size), and issues a comand to start capturing to the specified frame. The parameter of the **VIDIOCMCAPTURE** ioctl is **struct video_mmap**, shown here:

Field	Description
unsigned int frame	Frame number to capture to (for double or n-buffering)
int height	Frame height
int width	Frame width
unsigned int format	Sets the palette mode that should be used fill the frame. (see palette modes table)

The frame will be continously refreshed until **VIDIOCSYNC** ioctl comes. Whenever user program wants to start processing the frame data, it should first call **VIDIOCSYNC** to stop the capturing to this frame and to leave it in entire state. **VIDIOCSYNC** takes the frame number you are requesting to process as its argument. When the buffer is unmapped or all the frames in the buffer are told to stop refreshing, the capture ceases. While capturing to memory the driver will make a "best effort" attempt to capture to screen as well if requested. This normally means all frames that "miss" memory mapped capture will go to the display.

11 Miscellaneous

An ioctl exists to allow a device to obtain related devices if a driver has multiple components (for example video0 may not be associated with vbi0 which would cause an intercast display program to make a bad mistake). The **VIDIOCGUNIT** ioctl reports the unit numbers of the associated devices if any exist. The **video_unit** structure has the following fields:

video	Video capture device
vbi	VBI capture device
radio	Radio device
audio	Audio mixer
teletext	Teletext device