Linux Video4Linux2 API (v4l2)

Video4Linux or V4L is a collection of device drivers and an API for supporting readltime video capture on Linux systems. Various V4L drivers will create /dev/video* nodes in the filesystem that can be operated on for video capture.

This wiki page is created for use of v4l2 on Gateworks Rugged and Industrial Single Board Computers Made in the USA.

v4l2-ctl utility

The `v4l2-ctl` application from the `v4l-utils` package can be used to query or configure a v4l2 device.

For complete information, including output resolutions etc , use the following:

```
root@ventana:~# v4l2-ctl --device /dev/video3 --all # query all details about /dev/video3
```

See Also:

- GStreamer

video standards

Analog video v4l2 capture devices will support one or more of the various analog video standards used throughout the world. For example the ADV7180 supports NTSC, PAL, and SECAM standards.

You can use the Video4Linux API to determine if an input is detected and what mode by using the `v4l2-ctl --get-standard` command which reports one of the following:

- No supported input detected - Video Standard = 0x00ffffff
- NTSC - Video Standard = 0x000000ff
- PAL - Video Standard = 0x000000ff

Example:

- Show Analog Video status on GW51xx/GW52xx/GW53xx (/dev/video0):

```
v4l2-ctl --device /dev/video0 --get-standard
```

- Show Analog Video status on GW54xx (/dev/video1)

```
v4l2-ctl --device /devivideo0 --get-standard
```

video inputs

Some capture devices have multiple video inputs. The `v4l2-ctl` application can be used to get or set the current input.

Examples:

- show current inputs that /dev/video3 supports:

```
v4l2-ctl --device /dev/video3 --get-input
```

- select input1:

```
v4l2-ctl --device /dev/video3 --set-input=1
```

pixel formats, framesizes, and framerates

You can use `v4l2-ctl` to show what pixel formats a video device supports and select a desired format. Note that v4l2 devices support one or more pixel formats, and various resolutions and framerates for each format. You must ask what resolutions are supported for a particular format, and what framerates are supported for a particular resolution/format.

Examples:

- show formats that `/dev/video3` supports:

```
v4l2-ctl --device /dev/video3 --list-formats-ext
```

- show what resolutions `/dev/video3` supports for the UYVY pixel format:

```
v4l2-ctl --device /dev/video3 --list-framesizes=UYVY
```

- show what framerates `/dev/video3` supports for UYVY at 640x480:

```
v4l2-ctl --device /dev/video3 --list-
frameintervals=width=640,height=480,pixelformat=UYVY
```

- set format for `/dev/video0` to 1280x720 UYVY:

```
v4l2-ctl --device /dev/video0 --set-fmt-
video=width=1280,height=720,pixelformat=UYVY
```

It is not uncommon for a v4l2 driver to not properly support the enumeration of various formats, resolutions, and framerates. If you find that a device is not returning anything useful for `--list-formats` `--list-formats-ext` `--list-framesizes` `--list-framerates` then you can try `--all` to get the default configuration of the device.

Examples:

- the TDA1997x HDMI receiver on a GW54xx

```
root@ventana:~# v4l2-ctl --device /dev/video0 --all
Driver Info (not using libv4l2):
        Driver name   : mxc_v4l2
        Card type     :
        Bus info      :
        Driver version: 0.1.11
        Capabilities  : 0x05000005
                Video Capture
                Video Overlay
                Read/Write
                Streaming
Video input : 0 (CSI IC MEM: ok)
Video output: 0 (DISP3 BG)
Video Standard = 0x00000000
Format Video Capture:
        Width/Height  : 288/352
        Pixel Format  : 'UYVY'
        Field         : Any
        Bytes per Line: 432
        Size Image    : 152064
        Colorspace    : Unknown (00000000)
Format Video Overlay:
        Left/Top    : 0/0
        Width/Height: 160/160
        Field       : Any
        Chroma Key  : 0x00000000
        Global Alpha: 0x00
        Clip Count  : 0
        Clip Bitmap : No
Framebuffer Format:
        Capability    : Extern Overlay
        Flags         : Overlay Matches Capture/Output Size
        Width         : 0
        Height        : 0
        Pixel Format  : ''
Crop Capability Video Capture:
        Bounds      : Left 0, Top 0, Width 1920, Height 1080
        Default     : Left 0, Top 0, Width 1920, Height 1080
        Pixel Aspect: 0/0
Crop: Left 0, Top 0, Width 1920, Height 1080
Streaming Parameters Video Capture:
        Frames per second: 60.000 (60/1)
        Read buffers     : 0
Streaming Parameters Video Output:
        Frames per second: invalid (0/0)
        Write buffers    : 0
```

  - Note that the width/height does not reflect the resolution of the current video input source. In the above case a 1080p@60Hz input is attached. You can determine

the source resolution via the crop parameters.

- the ADV7180 Analog Video Decoder on a GW5400:

```
root@ventana:~# v4l2-ctl --device /dev/video1 --all
Driver Info (not using libv4l2):[ 1863.994275] ERROR:
unrecognized std! ffffff (PAL=ff, NTSC=b000

        Driver name   : mxc_v4l2
        Card type     :
        Bus info      :
        Driver version: 0.1.11
        Capabilities  : 0x05000005
                Video Capture
                Video Overlay
                Read/Write
                Streaming
Video input : 0 (CSI IC MEM: ok)
Video output: 0 (DISP3 BG)
Video Standard = 0x00ffffff
        PAL-B/B1/G/H/I/D/D1/K/M/N/Nc/60
        NTSC-M/M-JP/443/M-KR
        SECAM-B/D/G/H/K/K1/L/Lc
Format Video Capture:
        Width/Height  : 288/352
        Pixel Format  : 'YU12'
        Field         : Any
        Bytes per Line: 432
        Size Image    : 152064
        Colorspace    : Unknown (00000000)
Format Video Overlay:
        Left/Top    : 0/0
        Width/Height: 160/160
        Field       : Any
        Chroma Key  : 0x00000000
        Global Alpha: 0x00
        Clip Count  : 0
        Clip Bitmap : No
Framebuffer Format:
        Capability    : Extern Overlay
        Flags         : Overlay Matches Capture/Output Size
        Width         : 0
        Height        : 0
        Pixel Format  : ''
Crop Capability Video Capture:
        Bounds      : Left 0, Top 0, Width 720, Height 625
        Default     : Left 0, Top 0, Width 720, Height 625
        Pixel Aspect: 0/0
Crop: Left 0, Top 0, Width 720, Height 576
Streaming Parameters Video Capture:
        Frames per second: 30.000 (30/1)
        Read buffers     : 0
Streaming Parameters Video Output:
        Frames per second: invalid (0/0)
        Write buffers    : 0
```

capturing video frames

You can use `v4l2-ctl` to capture frames as well.

Examples:

- capture a single raw frame using mmap method:

```
v4l2-ctl --device /dev/video0 --stream-mmap --stream-
to=frame.raw --stream-count=1
```

Note that the convert tool from the ImageMagick package is useful for image format conversions as long as you know the specific details of the raw image.

Examples:

- convert a raw 640x480 16bit UYVY to png:

```
convert -size 640x480 -depth 16 uyvy:frame.raw frame.png
```

For an example v4l2 application for capturing frames see [https://linuxtv.org/downloads/v4l-dvb-apis/capture-example.html here].

For capturing video we recommend using more full-featured applications such as GStreamer

v4l2 application code

If you are coding an application that uses the v4l2 API the best examples are the v4l2-util application. You can find its source at https://git.linuxtv.org/v4l-utils.git/tree/

The API documentation can be found at:

- http://www.linuxtv.org/downloads/v4l-dvb-apis/

Note that there are many references online to what is considered the 'legacy' API used prior to the 3.0 kernel. Be sure to use the documentation above for the current version of the v4l2 API.

Examples:

- get/set/list video standard - v4l2-ctl-stds.cpp
- get/set/list video input - v4l2-ctl-io.cpp
- get/set/list video pixel format - v4l2-ctl-vidcap.cpp
- get/set/list video framesizes - v4l2-ctl-vidcap.cpp
- get/set/list video framerates - v4l2-ctl-vidcap.cpp
- video capture - v4l2-ctl-streaming.cpp v4l2-ctl-streaming.cpp

As a rule of thumb it is best to configure all aspects of a device within your code to eliminate any dependence on the initial state of a video device and/or requiring configuring a device outside of your application with something like `v4l2-ctl`. For completeness set the following:

- input (VIDIOC_S_INPUT)
- standard (VIDIOC_S_STD)
- format (VIDIOC_S_FMT) including width, height, pixel format, fields

- framerate (VIDIOC_S_PARM)

Be aware that the `ioctl` syscall can return a -1 result with `errno = EINTR` to indicate the syscall was interrupted in which case it should be tried again. Because of this you should consider providing a wrapper for ioctl such as:

```
static int xioctl(int fh, int request, void *arg)
{
        int r;

        do {
                r = ioctl(fh, request, arg);
        } while (-1 == r && EINTR == errno);

        return r;
}
```

IMX6 Ventana mxc_v4l2_capture driver

The Ventana product family based on the Freescale IMX6 System On Chip (SoC) has a Freescale provided capture driver (mxc_v4l2_capture) that is present in the Gateworks downstream vendor kernel used for the Yocto BSP.

This capture device driver implements two video inputs which represent different pipelines in the SoC:

- input0 (default) - CSI -> IC -> MEM: useful if you want to perform image manipulation using the IMX6 hardware blocks
- input1 - CSI -> MEM: raw image capture

If capturing frames be sure to skip the first frame as the CSI has not fully synchronized on its capture input.

To demonstrate a capture of a raw frame you can use the `v4l2_ctl` application:

```
# configure input 1 for CSI -> MEM (raw image capture)
v4l2-ctl --device /dev/video0 --set-input=1
# configure format
v4l2-ctl --device /dev/video0 --set-fmt-
video=width=1920,height=1080,pixelformat=UYVY
# capture 2nd frame
v4l2-ctl --device /dev/video0 --stream-mmap --stream-skip=1 --stream-
to=frame.raw --stream-count=1
# Use ImageMagick to convert it from raw 16-bit UYVY to PNG
convert -size 1920x1080 -depth 16 uyvy:frame.raw frame.png
```

Notes:

- The mxc_v4l2_capture driver does not support the enumeration ioctls necessary to query device capabilities (VIDIOC_ENUM_FMT, VIDIOC_ENUM_FRAMESIZES, VIDIOC_ENUM_FRAMEINTERVALS). The following format details show what is supported:
  - HDMI In via tda1997x (GW54xx/GW551x): V4L2_PIX_FMT_UYVY non-interlaced frames (even when using the yuv422bt656 capture mode the bt656 is converted

to non-interlaced frames by the IMX IC) with a resolution, framerate and colorspace dictated by the HDMI source.
    - CVBS In via adv7180: V4L2_PIX_FMT_UYVY non-interlaced frames (bt656 output is converted to non-interlaced frames by the IMX IC) with a resolution of either 720x480 (NTSC) or 720x576 (PAL) depending on the input source

Code Example:

- This example code will capture raw frames from the onboard video capture devices on the Ventana boards using the IMX6 CSI. It shows how to save the raw frame, process pixels (by counting the number of red pixels in the image), and convert it to a png via imagemagick

```
/*
 *  V4L2 video capture example
 *
 *  This program can be used and distributed without restrictions.
 *
 *      This program is provided with the V4L2 API
 * see http://linuxtv.org/docs.php for more information
 */

#include <byteswap.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <assert.h>
#include <fcntl.h>              /* low-level i/o */
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

#include <linux/videodev2.h>

#define CLEAR(x) memset(&(x), 0, sizeof(x))

struct buffer {
        void   *start;
        size_t  length;
};

static void errno_exit(const char *s)
{
        fprintf(stderr, "%s error %d, %s\n", s, errno, strerror(er
        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;
}

/* 16bit/pixel interleaved YUV */
static void process_image(const void *_p, struct v4l2_pix_format *
{
        const uint8_t *p = _p;
        int8_t u;
        uint8_t y1;
        int8_t v;
        uint8_t y2;
        int r, g, b, c, d ,e;
        int red, i, x, y;
        int size = fmt->sizeimage;

        printf("Processing Frame: %dx%d %c%c%c%c\n",
                fmt->width, fmt->height,
                (fmt->pixelformat >> 0) & 0xff,
                (fmt->pixelformat >> 8) & 0xff,
                (fmt->pixelformat >> 16) & 0xff,
                (fmt->pixelformat >> 24) & 0xff);

        red = 0;
        for (i = 0; i < size; i += 4) {
                u = p[i+0];
                y1 = p[i+1];
                v = p[i+2];
                y2 = p[i+3];

                u -= 128;
                v -= 128;

                r = y1 + v + (v>>2) + (v>>3) + (v>>5);
                g = y1 - ((u>>2) + (u>>4) + (u>>5))
                       - ((v>>1) + (v>>3) + (v>>4) + (v>>5));
                b = y1 + u + (u>>1) + (u>>2) + (u>>6);
                if (r > 100 && g < 60 && b < 60) red++;

                r = y2 + v + (v>>2) + (v>>3) + (v>>5);
                g = y2 - ((u>>2) + (u>>4) + (u>>5))
                       - ((v>>1) + (v>>3) + (v>>4) + (v>>5));
                b = y2 + u + (u>>1) + (u>>2) + (u>>6);
                if (r > 100 && g < 60 && b < 60) red++;

                /* describe pixels on first line every 250 pixels
                 * (colorbars)
                 */
                x = (i>>1) % fmt->width;
                y = (i>>1) / fmt->height;
                if (y == 0 && !(x % 250)) {
                        printf("[%4d,%4d] YUYV:0x%02x%02x%02x%02x%02x
                                        x,y,y1,(uint8_t)u,y2,(uint
                        printf("RGB:0x%02x%02x%02x\n", r,g,b);
                }
        }
        printf("red pixel count=%d\n", red);
}
```

```
static void save_frame(const char *path, const void *p, int size)
{
        int fd, rz;

        fd = open(path, O_WRONLY | O_CREAT | O_TRUNC, 0755);
        if (fd < 0)
                perror("open");
        else {
                rz = write(fd, p, size);
                printf("Wrote %d of %d bytes to %s\n", rz, size, p
                close(fd);
        }
}

int main(int argc, char **argv)
{
        static char *dev_name;
        int width, height;
        static int fd = -1;
        struct stat st;
        struct buffer *buffers;
        static unsigned int n_buffers;
        enum v4l2_buf_type type;
        struct v4l2_capability cap;
        struct v4l2_format fmt;
        struct v4l2_requestbuffers req;
        struct v4l2_streamparm parm;
        struct v4l2_input input;
        v4l2_std_id std_id;
        struct v4l2_buffer buf;
        unsigned int count;
        unsigned int i;
        char filename[32];

        /* parse args */
        if (argc < 5) {
                fprintf(stderr, "usage: %s <device> <width> <heigh
                        argv[0]);
                exit(1);
        }
        dev_name = argv[1];
        width = atoi(argv[2]);
        height = atoi(argv[3]);
        count = atoi(argv[4]);

        /* open device */
        fd = open(dev_name, O_RDWR | O_NONBLOCK, 0);
        if (-1 == fd) {
                fprintf(stderr, "Cannot open '%s': %d, %s\n",
                                dev_name, errno, strerror(errno));
                exit(EXIT_FAILURE);
        }

        /* get standard (wait for it to be locked onto a signal) *
        if (-1 == xioctl(fd, VIDIOC_G_STD, &std_id))
                perror("VIDIOC_G_STD");
        for (i = 0; std_id == V4L2_STD_ALL && i < 10; i++) {
                usleep(100000);
```

```
        xioctl(fd, VIDIOC_G_STD, &std_id);
}
/* set the standard to the detected standard (this is crit
if (std_id != V4L2_STD_UNKNOWN) {
        if (-1 == xioctl(fd, VIDIOC_S_STD, &std_id))
                perror("VIDIOC_S_STD");
        if (std_id & V4L2_STD_NTSC)
                printf("found NTSC TV decoder\n");
        if (std_id & V4L2_STD_SECAM)
                printf("found SECAM TV decoder\n");
        if (std_id & V4L2_STD_PAL)
                printf("found PAL TV decoder\n");
}

/* ensure device has video capture capability */
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);
}
if (!(cap.capabilities & V4L2_CAP_STREAMING)) {
        fprintf(stderr, "%s does not support streaming i/o
                        dev_name);
        exit(EXIT_FAILURE);
}

/* set video input */
CLEAR(input);
input.index = 1; /* IMX6 v4l2 driver: input1 is CSI<->MEM
if (-1 == xioctl(fd, VIDIOC_S_INPUT, &input))
        perror("VIDIOC_S_INPUT");

/* set framerate */
CLEAR(parm);
parm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (-1 == xioctl(fd, VIDIOC_S_PARM, &parm))
        perror("VIDIOC_S_PARM");

/* get framerate */
CLEAR(parm);
parm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (-1 == xioctl(fd, VIDIOC_G_PARM, &parm))
        perror("VIDIOC_G_PARM");

/* set format */
CLEAR(fmt);
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width       = width;
fmt.fmt.pix.height      = height;
```

```
        fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_UYVY;
        fmt.fmt.pix.field       = V4L2_FIELD_ANY;
        if (-1 == xioctl(fd, VIDIOC_S_FMT, &fmt))
                errno_exit("VIDIOC_S_FMT");

        /* get and display format */
        CLEAR(fmt);
        fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (-1 == xioctl(fd, VIDIOC_G_FMT, &fmt))
                errno_exit("VIDIOC_G_FMT");
        printf("%s: %dx%d %c%c%c%c %2.2ffps\n", dev_name,
                fmt.fmt.pix.width, fmt.fmt.pix.height,
                (fmt.fmt.pix.pixelformat >> 0) & 0xff,
                (fmt.fmt.pix.pixelformat >> 8) & 0xff,
                (fmt.fmt.pix.pixelformat >> 16) & 0xff,
                (fmt.fmt.pix.pixelformat >> 24) & 0xff,
                (float)parm.parm.capture.timeperframe.denominator
                (float)parm.parm.capture.timeperframe.numerator
                );

        /* request buffers */
        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 mapping\n", dev_name);
                        exit(EXIT_FAILURE);
                } else {
                        errno_exit("VIDIOC_REQBUFS");
                }
        }
        if (req.count < 2) {
                fprintf(stderr, "Insufficient buffer memory on %s\
                         dev_name);
                exit(EXIT_FAILURE);
        }

        /* allocate buffers */
        buffers = calloc(req.count, sizeof(*buffers));
        if (!buffers) {
                fprintf(stderr, "Out of memory\n");
                exit(EXIT_FAILURE);
        }

        /* mmap buffers */
        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");
        }

        /* queue buffers */
        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");
        }

        /* start capture */
        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
                errno_exit("VIDIOC_STREAMON");

        /* capture frame(s) (we throw away first incomplete frame
        for (i = 0; i < count + 1; i++) {
                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);
                        }

                        /* dequeue captured buffer */
```

```
                        CLEAR(buf);
                        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                        buf.memory = V4L2_MEMORY_MMAP;
                        if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf))
                                if (errno == EAGAIN)
                                        continue;
                                errno_exit("VIDIOC_DQBUF");
                        }
                        assert(buf.index < n_buffers);

                        /* skip first image as it may not be sync'
                        if (i > 0) {
                                process_image(buffers[buf.index].s
                                        &fmt.fmt.pix);
                                sprintf(filename, "frame%d.raw", i
                                save_frame(filename,
                                        buffers[buf.index].start,
                                        buf.bytesused);
                        }

                        /* queue buffer */
                        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                                errno_exit("VIDIOC_QBUF");

                        break;
                }
        }

        /* stop capture */
        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (-1 == xioctl(fd, VIDIOC_STREAMOFF, &type))
                errno_exit("VIDIOC_STREAMOFF");

        /* unmap and free buffers */
        for (i = 0; i < n_buffers; ++i)
                if (-1 == munmap(buffers[i].start, buffers[i].leng
                        errno_exit("munmap");
        free(buffers);

        /* close device */
        if (-1 == close(fd))
                errno_exit("close");

        fprintf(stderr, "\n");
        return 0;
}
```

Example usage:

- Capture HDMI input on GW5400 from a 1080P HDMI source:

```
$ ./capture /dev/video0 1920 1080 1
/dev/video0: 1920x1080 UYVY 60.00fps
Processing Frame: 1920x1080 UYVY
[   0,   0] YUYV:0xeb00eb00 RGB:0xebebeb
[ 250,   0] YUYV:0x9dad9d0d RGB:0xaeb409
[ 500,   0] YUYV:0x801b80ae RGB:0x0bb6ae
```

```
[ 750,   0] YUYV:0x6dc96dbb RGB:0x0ab50b
[1000,   0] YUYV:0x52365244 RGB:0xb111b0
red pixel count=259200
Wrote 4147200 of 4147200 bytes to frame.raw

./convert -size 1920x1080 -depth 16 uyvy:frame.raw frame.png #
convert to png via imagemagick
```

- Capture CVBS input on GW5400 from NTSC camera (720x480):

```
$ ./capture /dev/video1 720 480 1
/dev/video1: 720x520 UYVY 30.00fps
Processing Frame: 720x520 UYVY
[   0,   0] YUYV:0x1eff1c00 RGB:0x1c1f18
[ 250,   0] YUYV:0x1dfd1d01 RGB:0x1e2016
[ 500,   0] YUYV:0x28fa2aff RGB:0x26321e
red pixel count=27
Wrote 748800 of 748800 bytes to frame.raw

./convert -size 720x480 -depth 16 uyvy:frame.raw frame.png #
convert to png via imagemagick
```

Last modified on 05/21/2020 05:40:46 PM