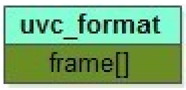
```
struct uvc_streaming {
	struct list_head list;	//uvc
	struct uvc_device *dev;		//uvc
	struct video_device *vdev;	//V4L2
	struct uvc_video_chain *chain;		//uvc
	atomic_t active;
	struct usb_interface *intf;	//usb
	int intfnum;	//usb
	__u16 maxpsize;
	struct uvc_streaming_header header;//uvc
	enum v4l2_buf_type type;	//V4L2
	unsigned int nformats;		//uvc
	struct uvc_format *format;	//uvc
	struct uvc_streaming_control ctrl;	//uvc
	struct uvc_format *cur_format;
	struct uvc_frame *cur_frame;
	struct mutex mutex;
	unsigned int frozen : 1;
	struct uvc_video_queue queue;		//uvc
	void (*decode) (struct urb *urb, struct uvc_streaming *video,struct uvc_buffer *buf);
	struct {
		__u8 header[256];
		unsigned int header_size;
		int skip_payload;
		__u32 payload_size;
		__u32 max_payload_size;
	} bulk;
	struct urb *urb[UVC_URBS];//urb
	char *urb_buffer[UVC_URBS];	//urb
	dma_addr_t urb_dma[UVC_URBS];//urb DMA
	unsigned int urb_size;
	__u32 sequence;
	__u8 last_fid;
};
```

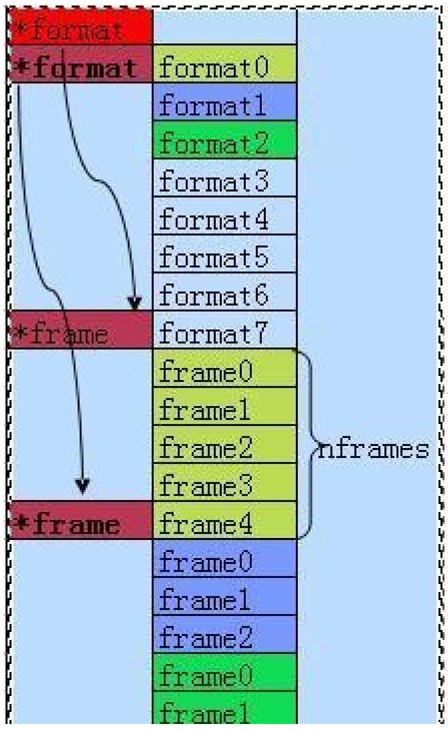
uvc_streaming
list
dev
vdev
intf
format[]

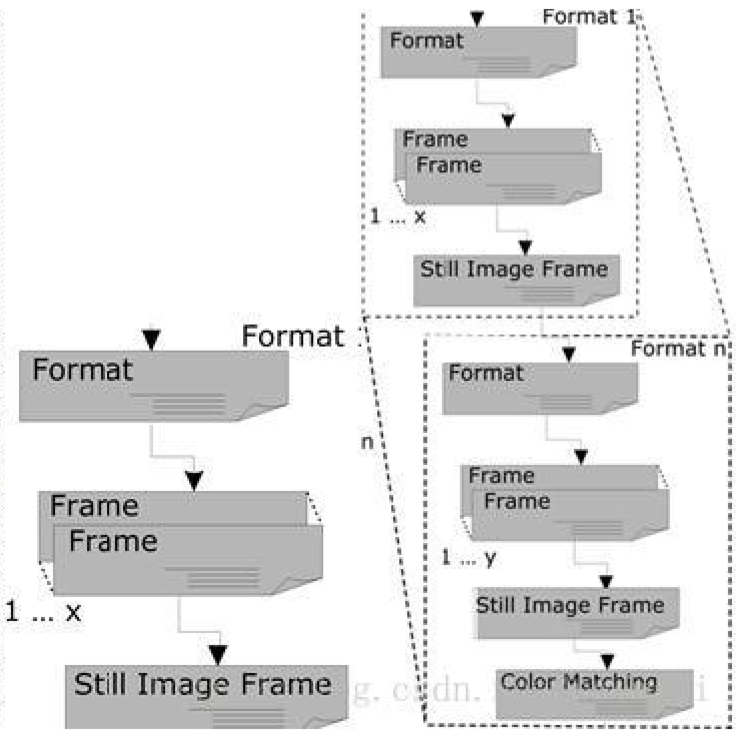
```
struct uvc_format {	//uvc
	__u8 type;	//
	__u8 index;	//
	__u8 bpp;	//bits per pixel
	__u8 colorspace;
	__u32 fcc;
	__u32 flags;
	char name[32];
	unsigned int nframes;
	struct uvc_frame *frame;
};
```



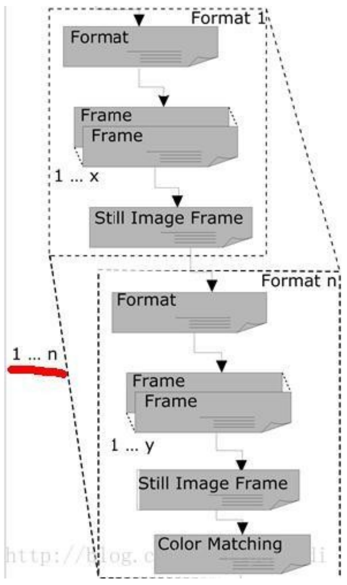
```
struct uvc_frame {	//uvc
	__u8  bFrameIndex;
	__u8  bmCapabilities;
	__u16 wWidth;
	__u16 wHeight;
	__u32 dwMinBitRate;
	__u32 dwMaxBitRate;
	__u32 dwMaxVideoFrameBufferSize;
	__u8  bFrameIntervalType;
	__u32 dwDefaultFrameInterval;
	__u32 *dwFrameInterval;
};
```

```
static int uvc_parse_streaming(struct uvc_device *dev,struct usb_interface *intf)
{
	struct uvc_streaming *streaming = NULL;   //uvc
	struct uvc_format *format;    //uvc
	struct uvc_frame *frame;      //uvc
	struct usb_host_interface *alts = &intf->altsetting[0];
usb_host_interface (Alt.Setting 0)
	unsigned char *_buffer, *buffer = alts->extra;
	int _buflen, buflen = alts->extralen;
	unsigned int nformats = 0, nframes = 0, nintervals = 0;
	unsigned int size, i, n, p;
	__u32 *interval;
	__u16 psize;
	int ret = -EINVAL;
	if (intf->cur_altsetting->desc.bInterfaceSubClass != UVC_SC_VIDEOSTREAMING) {
		uvc_trace(UVC_TRACE_DESCR, "device %d interface %d isn't a video streaming
interface\n", dev->udev->devnum,intf->altsetting[0].desc.bInterfaceNumber);
		return -EINVAL;
	}
	if (usb_driver_claim_interface(&uvc_driver.driver, intf, dev)) {
		uvc_trace(UVC_TRACE_DESCR, "device %d interface %d is already claimed\n",
dev->udev->devnum,intf->altsetting[0].desc.bInterfaceNumber);
		return -EINVAL;
	}
	streaming = kzalloc(sizeof *streaming, GFP_KERNEL);
	if (streaming == NULL) {
			usb_driver_release_interface(&uvc_driver.driver, intf);
			return -EINVAL;
	}
	mutex_init(&streaming->mutex);
	streaming->dev = dev;
	streaming->intf = usb_get_intf(intf);
	streaming->intfnum = intf->cur_altsetting->desc.bInterfaceNumber;
	/* The Pico iMage webcam has its class-specific interface descriptors after the endpoint
descriptors. */
	if (buflen == 0) {
		for (i = 0; i < alts->desc.bNumEndpoints; ++i) {
			struct usb_host_endpoint *ep = &alts->endpoint[i];
			if (ep->extralen == 0)
				continue;
			if (ep->extralen > 2 && ep->extra[1] == USB_DT_CS_INTERFACE) {
				uvc_trace(UVC_TRACE_DESCR, "trying extra data from endpoint
%u.\n", i);
				buffer = alts->endpoint[i].extra;
				buflen = alts->endpoint[i].extralen;
				break;
			}
		}
	}
	/* Skip the standard interface descriptors. */
	while (buflen > 2 && buffer[1] != USB_DT_CS_INTERFACE) {
		buflen -= buffer[0];
		buffer += buffer[0];
	}
	if (buflen <= 2) {
		uvc_trace(UVC_TRACE_DESCR, "no class-specific streaming interface descriptors
found.\n");
		goto error;
	}
	/* Parse the header descriptor. header*/
//Class-specific VS Interface Input Header Descriptor
	switch (buffer[2]) {	//bDescriptorSubtype
	case UVC_VS_OUTPUT_HEADER:
		streaming->type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
		size = 9;
		break;
	case UVC_VS_INPUT_HEADER:
		streaming->type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
		size = 13;
		break;
	default:
		uvc_trace(UVC_TRACE_DESCR, "device %d videostreaming interface %d
HEADER descriptor not found.\n", dev->udev->devnum,alts->desc.bInterfaceNumber);
		goto error;
	}
	p = buflen >= 4 ? buffer[3] : 0;		//bNumFormats uvc
	n = buflen >= size ? buffer[size-1] : 0;		//bControlSize
	if (buflen < size + p*n) {
		uvc_trace(UVC_TRACE_DESCR, "device %d videostreaming interface %d
HEADER descriptor is invalid.\n",dev->udev->devnum, alts->desc.bInterfaceNumber);
		goto error;
	}

	streaming->header.bNumFormats = p;
	streaming->header.bEndpointAddress = buffer[6];
	if (buffer[2] == UVC_VS_INPUT_HEADER) {
		streaming->header.bmInfo = buffer[7];
		streaming->header.bTerminalLink = buffer[8];
		streaming->header.bStillCaptureMethod = buffer[9];
(Method 1、Method 2、Method 3)
		streaming->header.bTriggerSupport = buffer[10];
		streaming->header.bTriggerUsage = buffer[11];
	}
	else {
		streaming->header.bTerminalLink = buffer[7];
	}
	streaming->header.bControlSize = n;
	streaming->header.bmaControls = kmemdup(&buffer[size], p * n,GFP_KERNEL);

	if (streaming->header.bmaControls == NULL) {
		ret = -ENOMEM;
		goto error;
	}
	buflen -= buffer[0];
	buffer += buffer[0];
	_buffer = buffer;
	_buflen = buflen;

/* Count the format and frame descriptors.*/
	while (_buflen > 2 && _buffer[1] == USB_DT_CS_INTERFACE) {
		switch (_buffer[2]) {
		case UVC_VS_FORMAT_UNCOMPRESSED:
		case UVC_VS_FORMAT_MJPEG:
		case UVC_VS_FORMAT_FRAME_BASED:
			nformats++;
			break;
		case UVC_VS_FORMAT_DV:
			/* DV format has no frame descriptor. We will create a dummy frame
descriptor with a dummy frame interval. */
			nformats++;
			nframes++;
			nintervals++;
			break;
		case UVC_VS_FORMAT_MPEG2TS:
		case UVC_VS_FORMAT_STREAM_BASED:
			uvc_trace(UVC_TRACE_DESCR, "device %d videostreaming interface %d
FORMAT %u is not supported.\n",dev->udev->devnum,alts->desc.bInterfaceNumber, _buffer[2]);
			break;
		case UVC_VS_FRAME_UNCOMPRESSED:
		case UVC_VS_FRAME_MJPEG:
					nframes++;
					if (_buflen > 25)
						nintervals += _buffer[25] ? _buffer[25] : 3;
					break;
				case UVC_VS_FRAME_FRAME_BASED:
					nframes++;
					if (_buflen > 21)
						nintervals += _buffer[21] ? _buffer[21] : 3;
					break;
				}
				_buflen -= _buffer[0];
				_buffer += _buffer[0];
		}
		if (nformats == 0) {
				uvc_trace(UVC_TRACE_DESCR, "device %d videostreaming interface %d has no
supported formats defined.\n",dev->udev->devnum, alts->desc.bInterfaceNumber);
				goto error;
		}

		size = nformats * sizeof *format + nframes * sizeof *frame+ nintervals * sizeof *interval;
		format = kzalloc(size, GFP_KERNEL);
		if (format == NULL) {
				ret = -ENOMEM;
				goto error;
		}
		frame = (struct uvc_frame *)&format[nformats];
		interval = (__u32 *)&frame[nframes];
		streaming->format = format;
		streaming->nformats = nformats;
/* Parse the format descriptors.*/
		while (buflen > 2 && buffer[1] == USB_DT_CS_INTERFACE) {
				switch (buffer[2]) {	//bDescriptorSubtype
				case UVC_VS_FORMAT_UNCOMPRESSED:
				case UVC_VS_FORMAT_MJPEG:
				case UVC_VS_FORMAT_DV:
				case UVC_VS_FORMAT_FRAME_BASED:
					format->frame = frame;
					ret = uvc_parse_format(dev, streaming, format,&interval, buffer, buflen);
					if (ret < 0)
						goto error;
					frame += format->nframes;
					format++;
					buflen -= ret;
					buffer += ret;
					continue;
```



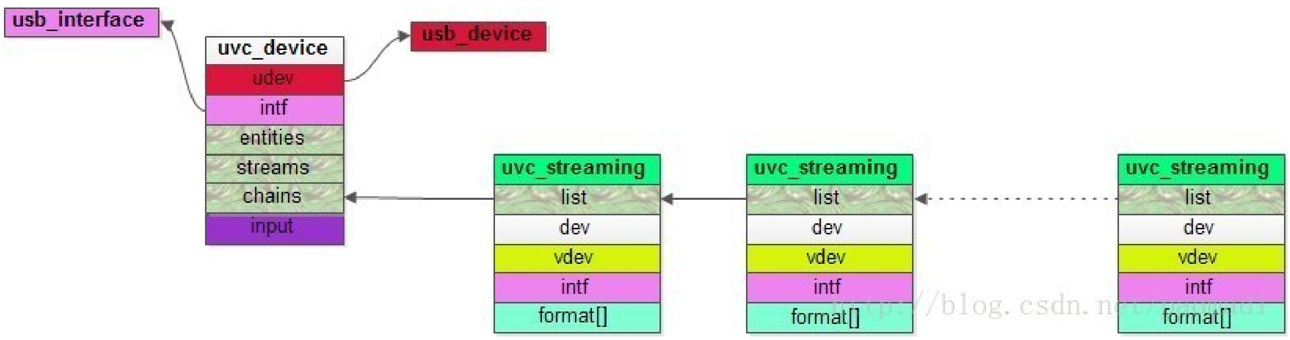


```
                default:
                        break;
                }
```



```
buflen -= buffer[0];
                buffer += buffer[0];
        }
        if (buflen)
                uvc_trace(UVC_TRACE_DESCR, "device %d videostreaming interface %d has %u
bytes of trailing descriptor garbage.\n",dev->udev->devnum, alts->desc.bInterfaceNumber, buflen);
        /* Parse the alternate settings to find the maximum bandwidth. */
        for (i = 0; i < intf->num_altsetting; ++i) {
                struct usb_host_endpoint *ep;
                alts = &intf->altsetting[i];
                ep = uvc_find_endpoint(alts,streaming->header.bEndpointAddress);
                if (ep == NULL)
                        continue;
                psize = le16_to_cpu(ep->desc.wMaxPacketSize);
                psize = (psize & 0x07ff) * (1 + ((psize >> 11) & 3));
                if (psize > streaming->maxpsize)
                        streaming->maxpsize = psize;
        }
        list_add_tail(&streaming->list, &dev->streams);
        return 0;
error:
        usb_driver_release_interface(&uvc_driver.driver, intf);
        usb_put_intf(intf);
        kfree(streaming->format);
        kfree(streaming->header.bmaControls);
        kfree(streaming);
        return ret;
}
```

list_add_tail(&streaming->list, &dev->streams);


usb_interface
uvc_device
udev
intf
entities
streams
chains
input
usb_device
uvc_streaming
list
dev
vdev
intf
format[]
uvc_streaming
list
dev
vdev
intf
format[]
uvc_streaming
list
dev
vdev
intf
format[]