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.

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:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\drivers\[linux-master][Documentation][userspace-api][media][drivers]uvcvideo.rst`, line 152)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        * 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:

System Message: WARNING/2 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\drivers\[linux-master][Documentation][userspace-api][media][drivers]uvcvideo.rst`, line 249)

Cannot analyze code. No Pygments lexer found for "none".

```
.. code-block:: none

        * 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
```