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 :c:type:`v4l2_subdev` struct (v4l2-subdev.h) was created.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 11); *backlink*

Unknown interpreted text role "c:type".

Each sub-device driver must have a :c:type:`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 :c:type:`v4l2_subdev` using :c:func:`v4l2_set_subdevdata`. That makes it easy to go from a :c:type:`v4l2_subdev` to the actual low-level bus-specific device data.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 15); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 15); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 15); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 15); *backlink*

Unknown interpreted text role "c:type".

You also need a way to go from the low-level struct to :c:type:`v4l2_subdev`. For the common i2c_client struct the i2c_set_clientdata() call is used to store a :c:type:`v4l2_subdev` pointer, for other buses you may have to use other methods.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 24); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 24); *backlink*

Unknown interpreted text role "c:type".

Bridges might also need to store per-subdev private data, such as a pointer to bridge-specific per-subdev private data. The :c:type:`v4l2_subdev` structure provides host private data for that purpose that can be accessed with :c:func:`v4l2_get_subdev_hostdata` and :c:func:`v4l2_set_subdev_hostdata`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 29); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 29); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 29); *backlink*

Unknown interpreted text role "c:func".

From the bridge driver perspective, you load the sub-device module and somehow obtain the :c:type:`v4l2_subdev` pointer. For i2c devices this is easy: you call `i2c_get_clientdata()`. For other buses something similar needs to be done. Helper functions exist for sub-devices on an I2C bus that do most of this tricky work for you.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 34); *backlink*

Unknown interpreted text role "c:type".

Each :c:type:`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.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 40); *backlink*

Unknown interpreted text role "c:type".

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 :c:type:`v4l2_subdev` struct using:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 90); *backlink*

Unknown interpreted text role "c:type".

:c:func:`v4l2_subdev_init <v4l2_subdev_init>` (:c:type:`sd <v4l2_subdev>`, &:c:type:`ops <v4l2_subdev_ops>`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 92); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 92); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 92); *backlink*

Unknown interpreted text role "c:type".

Afterwards you need to initialize :c:type:`sd <v4l2_subdev>`->name with a unique name and set the module owner. This is done for you if you use the i2c helper functions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 96); *backlink*

Unknown interpreted text role "c:type".

If integration with the media framework is needed, you must initialize the :c:type:`media_entity` struct embedded in the :c:type:`v4l2_subdev` struct (entity field) by calling :c:func:`media_entity_pads_init`, if the entity has pads:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 100); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 100); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 100); *backlink*

Unknown interpreted text role "c:func".

```
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 a sub-device driver implements sink pads, the subdev driver may set the link_validate field in :c:type:`v4l2_subdev_pad_ops` to provide its own link validation function. For every link in the pipeline, the link_validate pad operation of the sink end of the link is called. In both cases the driver is still responsible for validating the correctness of the format configuration between sub-devices and video nodes.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 125); *backlink***
>
> Unknown interpreted text role "c:type".

If link_validate op is not set, the default function :c:func:`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.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 132); *backlink***
>
> Unknown interpreted text role "c:func".

Subdev registration

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 :c:type:`v4l2_subdev` with the v4l2_device:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 160); *backlink***
>
> Unknown interpreted text role "c:type".

:c:func:`v4l2_device_register_subdev <v4l2_device_register_subdev>` (:c:type:`v4l2_dev <v4l2_device>`, :c:type:`sd <v4l2_subdev>`).

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 163); *backlink***
>
> Unknown interpreted text role "c:func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 163); *backlink***
>
> Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 163); *backlink*

Unknown interpreted text role "c:type".

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 :c:type:`v4l2_device`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 166); *backlink*

Unknown interpreted text role "c:type".

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:

:c:func:`v4l2_device_unregister_subdev <v4l2_device_unregister_subdev>` (:c:type:`sd <v4l2_subdev>`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 175); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 175); *backlink*

Unknown interpreted text role "c:type".

Afterwards the subdev module can be unloaded and :c:type:`sd <v4l2_subdev>`->dev == `NULL`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 179); *backlink*

Unknown interpreted text role "c:type".

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 :c:func:`v4l2_async_register_subdev` function. Unregistration is performed using the :c:func:`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.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 182); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 182); *backlink*

Unknown interpreted text role "c:func".

Bridge drivers in turn have to register a notifier object. This is performed using the :c:func:`v4l2_async_nf_register` call. To unregister the notifier the driver has to call :c:func:`v4l2_async_nf_unregister`. The former of the two functions takes two arguments: a pointer to struct :c:type:`v4l2_device` and a pointer to struct :c:type:`v4l2_async_notifier`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 193); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 193); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 193); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 193); *backlink*

Unknown interpreted text role "c:type".

Before registering the notifier, bridge drivers must do two things: first, the notifier must be initialized using the :c:func:`v4l2_async_nf_init`. Second, bridge drivers can then begin to form a list of subdevice descriptors that the bridge device needs for its operation. Several functions are available to add subdevice descriptors to a notifier, depending on the type of device and the needs of the driver.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 200); *backlink*

Unknown interpreted text role "c:func".

:c:func:`v4l2_async_nf_add_fwnode_remote` and :c:func:`v4l2_async_nf_add_i2c` are for bridge and ISP drivers for registering their async sub-devices with the notifier.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 207); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 207); *backlink*

Unknown interpreted text role "c:func".

:c:func:`v4l2_async_register_subdev_sensor` is a helper function for sensor drivers registering their own async sub-device, but it also registers a notifier and further registers async sub-devices for lens and flash devices found in firmware. The notifier for the sub-device is unregistered with the async sub-device.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 211); *backlink*

Unknown interpreted text role "c:func".

These functions allocate an async sub-device descriptor which is of type struct :c:type:`v4l2_async_subdev` embedded in a driver-specific struct. The &struct :c:type:`v4l2_async_subdev` shall be the first member of this struct:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 217); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media)v4l2-subdev.rst`, line 217); *backlink*

Unknown interpreted text role "c:type".

```
struct my_async_subdev {
        struct v4l2_async_subdev asd;
        ...
};

struct my_async_subdev *my_asd;
struct fwnode_handle *ep;

...

my_asd = v4l2_async_nf_add_fwnode_remote(&notifier, ep,
                                         struct my_async_subdev);
fwnode_handle_put(ep);

if (IS_ERR(asd))
        return PTR_ERR(asd);
```

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.

Calling subdev operations

The advantage of using :c:type:`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.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media)v4l2-subdev.rst`, line 249); *backlink*

Unknown interpreted text role "c:type".

Once the subdev has been registered 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 do the right `NULL` pointer checks and returns `-ENODEV` if :c:type:`sd <v4l2_subdev>` is `NULL`, `-ENOIOCTLCMD` if either :c:type:`sd <v4l2_subdev>`->core or :c:type:`sd <v4l2_subdev>`->core->g_std is `NULL`, or the actual result of the :c:type:`sd <v4l2_subdev>`->ops->core->g_std ops.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media)v4l2-subdev.rst`, line 268); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media)v4l2-subdev.rst`, line 268); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media)v4l2-subdev.rst`, line 268); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 268); *backlink*

Unknown interpreted text role "c:type".

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 :c:type:`sd <v4l2_subdev>`->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.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 289); *backlink*

Unknown interpreted text role "c:type".

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.

V4L2 sub-device userspace API

Bridge drivers traditionally expose one or multiple video nodes to userspace, and control subdevices through the :c:type:`v4l2_subdev_ops` operations in response to video node operations. This hides the complexity of the underlying hardware from applications. For complex devices, finer-grained control of the device than what the video nodes offer may be required. In those cases, bridge drivers that implement :ref:`the media controller API <media_controller>` may opt for making the subdevice operations directly accessible from userpace.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 312); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 312); *backlink*

Unknown interpreted text role "ref".

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 :c:type:`v4l2_device` driver can create device nodes for all registered sub-devices marked with `V4L2_SUBDEV_FL_HAS_DEVNODE` by calling :c:func:`v4l2_device_register_subdev_nodes`. Those device nodes will be automatically removed when sub-devices are unregistered.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 324); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 324); *backlink*

Unknown interpreted text role "c:func".

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_FL_HAS_EVENTS` :c:type:`v4l2_subdev`.flags before registering the sub-device. After registration events can be queued as usual on the :c:type:`v4l2_subdev`.devnode device node.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 355); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 355); *backlink*

Unknown interpreted text role "c:type".

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.

Read-only sub-device userspace API

Bridge drivers that control their connected subdevices through direct calls to the kernel API realized by :c:type:`v4l2_subdev_ops` structure do not usually want userspace to be able to change the same parameters through the subdevice device node and thus do not usually register any.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 371); *backlink*

Unknown interpreted text role "c:type".

It is sometimes useful to report to userspace the current subdevice configuration through a read-only API, that does not permit applications to change to the device parameters but allows interfacing to the subdevice device node to inspect them.

For instance, to implement cameras based on computational photography, userspace needs to know the detailed camera sensor configuration (in terms of skipping, binning, cropping and scaling) for each supported output resolution. To support such use cases, bridge drivers may expose the subdevice operations to userspace through a read-only API.

To create a read-only device node for all the subdevices registered with the `V4L2_SUBDEV_FL_HAS_DEVNODE` set, the :c:type:`v4l2_device` driver should call :c:func:`v4l2_device_register_ro_subdev_nodes`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 387); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media) v4l2-subdev.rst, line 387); *backlink*

Unknown interpreted text role "c:func".

Access to the following ioctls for userspace applications is restricted on sub-device device nodes registered with :c:func:`v4l2_device_register_ro_subdev_nodes`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media) v4l2-subdev.rst, line 391); *backlink*

Unknown interpreted text role "c:func".

`VIDIOC_SUBDEV_S_FMT`, `VIDIOC_SUBDEV_S_CROP`, `VIDIOC_SUBDEV_S_SELECTION`:

These ioctls are only allowed on a read-only subdevice device node for the :ref:`V4L2_SUBDEV_FORMAT_TRY <v4l2-subdev-format-whence>` formats and selection rectangles.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media) v4l2-subdev.rst, line 399); *backlink*

Unknown interpreted text role "ref".

`VIDIOC_SUBDEV_S_FRAME_INTERVAL`, `VIDIOC_SUBDEV_S_DV_TIMINGS`, `VIDIOC_SUBDEV_S_STD`:

These ioctls are not allowed on a read-only subdevice node.

In case the ioctl is not allowed, or the format to modify is set to `V4L2_SUBDEV_FORMAT_ACTIVE`, the core returns a negative error code and the errno variable is set to `-EPERM`.

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 :c:type:`v4l2_subdev` support to an I2C driver is to embed the :c:type:`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 :c:type:`v4l2_subdev` directly.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media) v4l2-subdev.rst, line 419); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media) v4l2-subdev.rst, line 419); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master) (Documentation) (driver-api) (media) v4l2-subdev.rst, line 419); *backlink*

Unknown interpreted text role "c:type".

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 :c:type:`v4l2_subdev` struct as follows:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 434); *backlink*

Unknown interpreted text role "c:type".

```
v4l2_i2c_subdev_init(&state->sd, client, subdev_ops);
```

This function will fill in all the fields of :c:type:`v4l2_subdev` ensure that the :c:type:`v4l2_subdev` and i2c_client both point to one another.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 440); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 440); *backlink*

Unknown interpreted text role "c:type".

You should also add a helper inline function to go from a :c:type:`v4l2_subdev` pointer to a chipname_state struct:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 443); *backlink*

Unknown interpreted text role "c:type".

```
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 :c:type:`v4l2_subdev` struct to the `i2c_client` struct:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 453); *backlink*

Unknown interpreted text role "c:type".

```
struct i2c_client *client = v4l2_get_subdevdata(sd);
```

And this to go from an `i2c_client` to a :c:type:`v4l2_subdev` struct:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 460); *backlink*

Unknown interpreted text role "c:type".

```
struct v4l2_subdev *sd = i2c_get_clientdata(client);
```

Make sure to call :c:func:`v4l2_device_unregister_subdev`(:c:type:`sd <v4l2_subdev>`) 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.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 466); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 466); *backlink*

Unknown interpreted text role "c:type".

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 :c:func:`v4l2_device_unregister_subdev`(:c:type:`sd <v4l2_subdev>`) from the `remove()` callback ensures that this is always done correctly.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 472); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 472); *backlink*

Unknown interpreted text role "c:type".

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 :c:func:`i2c_new_client_device` with the given `i2c_adapter` and chip/address arguments. If all goes well, then it registers the subdev with the v4l2_device.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 487); *backlink*

Unknown interpreted text role "c:func".

You can also use the last argument of :c:func:`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.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 492); *backlink*

Unknown interpreted text role "c:func".

Both functions return `NULL` if something went wrong.

Note that the chipid you pass to :c:func:`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.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 499); *backlink*

Unknown interpreted text role "c:func".

There are one more helper function:

:c:func:`v4l2_i2c_new_subdev_board` uses an :c:type:`i2c_board_info` struct which is passed to the i2c driver and replaces the irq, platform_data and addr arguments.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 509); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 509); *backlink*

Unknown interpreted text role "c:type".

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 :c:func:`v4l2_i2c_new_subdev` function will call :c:func:`v4l2_i2c_new_subdev_board`, internally filling a :c:type:`i2c_board_info` structure using the `client_type` and the `addr` to fill it.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 516); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 516); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 516); *backlink*

Unknown interpreted text role "c:type".

V4L2 sub-device functions and data structures

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)v4l2-subdev.rst, line 524)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/media/v4l2-subdev.h
```