Media Controller devices

Media Controller

The media controller userspace API is documented in :ref:`the Media Controller uAPI book <media_controller>`. This document focus on the kernel-side implementation of the media framework.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 9); *backlink*

Unknown interpreted text role "ref".

Abstract media device model

Discovering a device internal topology, and configuring it at runtime, is one of the goals of the media framework. To achieve this, hardware devices are modelled as an oriented graph of building blocks called entities connected through pads.

An entity is a basic media hardware building block. It can correspond to a large variety of logical blocks such as physical hardware devices (CMOS sensor for instance), logical hardware devices (a building block in a System-on-Chip image processing pipeline), DMA channels or physical connectors.

A pad is a connection endpoint through which an entity can interact with other entities. Data (not restricted to video) produced by an entity flows from the entity's output to one or more entity inputs. Pads should not be confused with physical pins at chip boundaries.

A link is a point-to-point oriented connection between two pads, either on the same entity or on different entities. Data flows from a source pad to a sink pad.

Media device

A media device is represented by a struct media_device instance, defined in `include/media/media-device.h`. Allocation of the structure is handled by the media device driver, usually by embedding the :c:type:`media_device` instance in a larger driver-specific structure.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 39); *backlink*

Unknown interpreted text role "c:type".

Drivers register media device instances by calling :c:func:`__media_device_register()` via the macro `media_device_register()` and unregistered by calling :c:func:`media_device_unregister()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 45); *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)mc-core.rst, line 45); *backlink*

Unknown interpreted text role "c:func".

Entities

Entities are represented by a struct media_entity instance, defined in `include/media/media-entity.h`. The structure is usually embedded into a higher-level structure, such as :c:type:`v4l2_subdev` or :c:type:`video_device` instances, although drivers can allocate entities directly.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 52); *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)mc-core.rst, line 52); *backlink*

Unknown interpreted text role "c:type".

Drivers initialize entity pads by calling :c:func:`media_entity_pads_init()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 58); *backlink*

Unknown interpreted text role "c:func".

Drivers register entities with a media device by calling :c:func:`media_device_register_entity()` and unregistered by calling :c:func:`media_device_unregister_entity()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 61); *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)mc-core.rst, line 61); *backlink*

Unknown interpreted text role "c:func".

Interfaces

Interfaces are represented by a struct media_interface instance, defined in `include/media/media-entity.h`. Currently, only one type of interface is defined: a device node. Such interfaces are represented by a struct media_intf_devnode.

Drivers initialize and create device node interfaces by calling :c:func:`media_devnode_create()` and remove them by calling: :c:func:`media_devnode_remove()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 75); *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)mc-core.rst, line 75); *backlink*

Unknown interpreted text role "c:func".

Pads

Pads are represented by a struct media_pad instance, defined in `include/media/media-entity.h`. Each entity stores its pads in a pads array managed by the entity driver. Drivers usually embed the array in a driver-specific structure.

Pads are identified by their entity and their 0-based index in the pads array.

Both information are stored in the struct media_pad, making the struct media_pad pointer the canonical way to store and pass link references.

Pads have flags that describe the pad capabilities and state.

`MEDIA_PAD_FL_SINK` indicates that the pad supports sinking data. `MEDIA_PAD_FL_SOURCE` indicates that the pad supports sourcing data.

Note

One and only one of `MEDIA_PAD_FL_SINK` or `MEDIA_PAD_FL_SOURCE` must be set for each pad.

Links

Links are represented by a struct media_link instance, defined in `include/media/media-entity.h`. There are two types of links:

1. pad to pad links:

Associate two entities via their PADs. Each entity has a list that points to all links originating at or targeting any of its pads. A given link is thus stored twice, once in the source entity and once in the target entity.

Drivers create pad to pad links by calling: :c:func:`media_create_pad_link()` and remove with :c:func:`media_entity_remove_links()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 117); *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)mc-core.rst, line 117); *backlink*

Unknown interpreted text role "c:func".

2. interface to entity links:

Associate one interface to a Link.

Drivers create interface to entity links by calling: :c:func:`media_create_intf_link()` and remove with :c:func:`media_remove_intf_links()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 125); *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)mc-core.rst, line 125); *backlink*

Unknown interpreted text role "c:func".

Note

Links can only be created after having both ends already created.

Links have flags that describe the link capabilities and state. The valid values are described at :c:func:`media_create_pad_link()` and :c:func:`media_create_intf_link()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 133); *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)mc-core.rst, line 133); *backlink*

Unknown interpreted text role "c:func".

Graph traversal

The media framework provides APIs to iterate over entities in a graph.

To iterate over all entities belonging to a media device, drivers can use the media_device_for_each_entity macro, defined in

include/media/media-device.h.

```
struct media_entity *entity;

media_device_for_each_entity(entity, mdev) {
// entity will point to each entity in turn
...
}
```

Drivers might also need to iterate over all entities in a graph that can be reached only through enabled links starting at a given entity. The media framework provides a depth-first graph traversal API for that purpose.

> **Note**
>
> Graphs with cycles (whether directed or undirected) are **NOT** supported by the graph traversal API. To prevent infinite loops, the graph traversal code limits the maximum depth to `MEDIA_ENTITY_ENUM_MAX_DEPTH`, currently defined as 16.

Drivers initiate a graph traversal by calling :c:func:`media_graph_walk_start()`

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 166); *backlink***
>
> Unknown interpreted text role "c:func".

The graph structure, provided by the caller, is initialized to start graph traversal at the given entity.

Drivers can then retrieve the next entity by calling :c:func:`media_graph_walk_next()`

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 172); *backlink***
>
> Unknown interpreted text role "c:func".

When the graph traversal is complete the function will return `NULL`.

Graph traversal can be interrupted at any moment. No cleanup function call is required and the graph structure can be freed normally.

Helper functions can be used to find a link between two given pads, or a pad connected to another pad through an enabled link :c:func:`media_entity_find_link()` and :c:func:`media_entity_remote_pad()`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 180); *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)mc-core.rst, line 180); *backlink***
>
> Unknown interpreted text role "c:func".

Use count and power handling

Due to the wide differences between drivers regarding power management needs, the media controller does not implement power management. However, the struct media_entity includes a `use_count` field that media drivers can use to track the number of users of every entity for power management needs.

The :c:type:`media_entity<media_entity>`.`use_count` field is owned by media drivers and must not be touched by entity drivers. Access to the field must be protected by the :c:type:`media_device`.`graph_mutex` lock.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 195); *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)mc-core.rst, line 195); *backlink*

Unknown interpreted text role "c:type".

Links setup

Link properties can be modified at runtime by calling :c:func:`media_entity_setup_link()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 203); *backlink*

Unknown interpreted text role "c:func".

Pipelines and media streams

When starting streaming, drivers must notify all entities in the pipeline to prevent link states from being modified during streaming by calling :c:func:`media_pipeline_start()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 209); *backlink*

Unknown interpreted text role "c:func".

The function will mark all entities connected to the given entity through enabled links, either directly or indirectly, as streaming.

The struct media_pipeline instance pointed to by the pipe argument will be stored in every entity in the pipeline. Drivers should embed the struct media_pipeline in higher-level pipeline structures and can then access the pipeline through the struct media_entity pipe field.

Calls to :c:func:`media_pipeline_start()` can be nested. The pipeline pointer must be identical for all nested calls to the function.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 223); *backlink*

Unknown interpreted text role "c:func".

:c:func:`media_pipeline_start()` may return an error. In that case, it will clean up any of the changes it did by itself.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 226); *backlink*

Unknown interpreted text role "c:func".

When stopping the stream, drivers must notify the entities with :c:func:`media_pipeline_stop()`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 229); *backlink*

Unknown interpreted text role "c:func".

If multiple calls to :c:func:`media_pipeline_start()` have been made the same number of :c:func:`media_pipeline_stop()` calls are required to stop streaming. The :c:type:`media_entity`.`pipe` field is reset to `NULL` on the last nested stop call.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 232); *backlink*

Unknown interpreted text role "c:func".

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)mc-core.rst, line 232); *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)mc-core.rst, line 232); *backlink*

Unknown interpreted text role "c:type".

Link configuration will fail with `-EBUSY` by default if either end of the link is a streaming entity. Links that can be modified while streaming must be marked with the `MEDIA_LNK_FL_DYNAMIC` flag.

If other operations need to be disallowed on streaming entities (such as changing entities configuration parameters) drivers can explicitly check the media_entity stream_count field to find out if an entity is streaming. This operation must be done with the media_device graph_mutex held.

Link validation

Link validation is performed by :c:func:`media_pipeline_start()` for any entity which has sink pads in the pipeline. The :c:type:`media_entity`.`link_validate()` callback is used for that purpose. In `link_validate()` callback, entity driver should check that the properties of the source pad of the connected entity and its own sink pad match. It is up to the type of the entity (and in the end, the properties of the hardware) what matching actually means.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 250); *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)mc-core.rst, line 250); *backlink*

Unknown interpreted text role "c:type".

Subsystems should facilitate link validation by providing subsystem specific helper functions to provide easy access for commonly needed information, and in the end provide a way to use driver-specific callbacks.

Media Controller Device Allocator API

When the media device belongs to more than one driver, the shared media device is allocated with the shared struct device as the key for look ups.

The shared media device should stay in registered state until the last driver unregisters it. In addition, the media device should be released when all the references are released. Each driver gets a reference to the media device during probe, when it allocates the media device. If media device is already allocated, the allocate API bumps up the refcount and returns the existing media device. The driver puts the reference back in its disconnect routine when it calls :c:func:`media_device_delete()`.

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

Unknown interpreted text role "c:func".

The media device is unregistered and cleaned up from the kref put handler to ensure that the media device stays in registered state until the last driver unregisters the media device.

Driver Usage

Drivers should use the appropriate media-core routines to manage the shared media device life-time handling the two states: 1. allocate -> register -> delete 2. get reference to already registered device -> delete

call :c:func:`media_device_delete()` routine to make sure the shared media device delete is handled correctly.

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

(media)mc-core.rst, line 287); *backlink*

Unknown interpreted text role "c:func".

driver probe: Call :c:func:`media_device_usb_allocate()` to allocate or get a reference Call :c:func:`media_device_register()`, if media devnode isn't registered

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 290); *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)mc-core.rst, line 290); *backlink*

Unknown interpreted text role "c:func".

driver disconnect: Call :c:func:`media_device_delete()` to free the media_device. Freeing is handled by the kref put handler.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 294); *backlink*

Unknown interpreted text role "c:func".

API Definitions

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 301)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/media/media-device.h
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 303)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/media/media-devnode.h
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 305)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/media/media-entity.h
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 307)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/media/media-request.h
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\media\(linux-master)(Documentation)(driver-api)(media)mc-core.rst, line 309)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/media/media-dev-allocator.h
```