Request API

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 2)

Unknown directive type "c:namespace".

```
.. c:namespace:: MC
```

The Request API has been designed to allow V4L2 to deal with requirements of modern devices (stateless codecs, complex camera pipelines, ...) and APIs (Android Codec v2). One such requirement is the ability for devices belonging to the same pipeline to reconfigure and collaborate closely on a per-frame basis. Another is support of stateless codecs, which require controls to be applied to specific frames (aka 'per-frame controls') in order to be used efficiently.

While the initial use-case was V4L2, it can be extended to other subsystems as well, as long as they use the media controller.

Supporting these features without the Request API is not always possible and if it is, it is terribly inefficient: user-space would have to flush all activity on the media pipeline, reconfigure it for the next frame, queue the buffers to be processed with that configuration, and wait until they are all available for dequeuing before considering the next frame. This defeats the purpose of having buffer queues since in practice only one buffer would be queued at a time.

The Request API allows a specific configuration of the pipeline (media controller topology + configuration for each media entity) to be associated with specific buffers. This allows user-space to schedule several tasks ("requests") with different configurations in advance, knowing that the configuration will be applied when needed to get the expected result. Configuration values at the time of request completion are also available for reading.

General Usage

The Request API extends the Media Controller API and cooperates with subsystem-specific APIs to support request usage. At the Media Controller level, requests are allocated from the supporting Media Controller device node. Their life cycle is then managed through the request file descriptors in an opaque way. Configuration data, buffer handles and processing results stored in requests are accessed through subsystem-specific APIs extended for request support, such as V4L2 APIs that take an explicit `request_fd` parameter.

Request Allocation

User-space allocates requests using :ref:`MEDIA_IOC_REQUEST_ALLOC` for the media device node. This returns a file descriptor representing the request. Typically, several such requests will be allocated.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 48); *backlink*

Unknown interpreted text role "ref".

Request Preparation

Standard V4L2 ioctls can then receive a request file descriptor to express the fact that the ioctl is part of said request, and is not to be applied immediately. See :ref:`MEDIA_IOC_REQUEST_ALLOC` for a list of ioctls that support this. Configurations set with a `request_fd` parameter are stored instead of being immediately applied, and buffers queued to a request do not enter the regular buffer queue until the request itself is queued.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 55); *backlink*

Unknown interpreted text role "ref".

Request Submission

Once the configuration and buffers of the request are specified, it can be queued by calling :ref:`MEDIA_REQUEST_IOC_QUEUE` on the request file descriptor. A request must contain at least one buffer, otherwise `ENOENT` is returned. A queued request cannot be modified anymore.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 65); *backlink*

Unknown interpreted text role "ref".

Caution!

For :ref:`memory-to-memory devices <mem2mem>` you can use requests only for output buffers, not for capture buffers. Attempting to add a capture buffer to a request will result in an `EBADR` error.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 71); *backlink*

Unknown interpreted text role "ref".

If the request contains configurations for multiple entities, individual drivers may synchronize so the requested pipeline's topology is applied before the buffers are processed. Media controller drivers do a best effort implementation since perfect atomicity may not be possible due to hardware limitations.

Caution!

It is not allowed to mix queuing requests with directly queuing buffers: whichever method is used first locks this in place until :ref:`VIDIOC_STREAMOFF <VIDIOC_STREAMON>` is called or the device is :ref:`closed <func-close>`. Attempts to directly queue a buffer when earlier a buffer was queued via a request or vice versa will result in an `EBUSY` error.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 82); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 82); *backlink*

Unknown interpreted text role "ref".

Controls can still be set without a request and are applied immediately, regardless of whether a request is in use or not.

Caution!

Setting the same control through a request and also directly can lead to undefined behavior!

User-space can :c:func:`poll()` a request file descriptor in order to wait until the request completes. A request is considered complete once all its associated buffers are available for dequeuing and all the associated controls have been updated with the values at the time of completion. Note that user-space does not need to wait for the request to complete to dequeue its buffers: buffers that are available halfway through a request can be dequeued independently of the request's state.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 97); *backlink*

Unknown interpreted text role "c:func".

A completed request contains the state of the device after the request was executed. User-space can query that state by calling :ref:`ioctl VIDIOC_G_EXT_CTRLS <VIDIOC_G_EXT_CTRLS>` with the request file descriptor. Calling :ref:`ioctl VIDIOC_G_EXT_CTRLS <VIDIOC_G_EXT_CTRLS>` for a request that has been queued but not yet completed will return

EBUSY since the control values might be changed at any time by the driver while the request is in flight.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 105); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 105); *backlink*

Unknown interpreted text role "ref".

Recycling and Destruction

Finally, a completed request can either be discarded or be reused. Calling :c:func:`close()` on a request file descriptor will make that file descriptor unusable and the request will be freed once it is no longer in use by the kernel. That is, if the request is queued and then the file descriptor is closed, then it won't be freed until the driver completed the request.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 118); *backlink*

Unknown interpreted text role "c:func".

The :ref:`MEDIA_REQUEST_IOC_REINIT` will clear a request's state and make it available again. No state is retained by this operation: the request is as if it had just been allocated.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 125); *backlink*

Unknown interpreted text role "ref".

Example for a Codec Device

For use-cases such as :ref:`codecs <mem2mem>`, the request API can be used to associate specific controls to be applied by the driver for the OUTPUT buffer, allowing user-space to queue many such buffers in advance. It can also take advantage of requests' ability to capture the state of controls when the request completes to read back information that may be subject to change.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 132); *backlink*

Unknown interpreted text role "ref".

Put into code, after obtaining a request, user-space can assign controls and one OUTPUT buffer to it:

```
struct v4l2_buffer buf;
struct v4l2_ext_controls ctrls;
int req_fd;
...
if (ioctl(media_fd, MEDIA_IOC_REQUEST_ALLOC, &req_fd))
        return errno;
...
ctrls.which = V4L2_CTRL_WHICH_REQUEST_VAL;
ctrls.request_fd = req_fd;
if (ioctl(codec_fd, VIDIOC_S_EXT_CTRLS, &ctrls))
        return errno;
...
buf.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
buf.flags |= V4L2_BUF_FLAG_REQUEST_FD;
buf.request_fd = req_fd;
if (ioctl(codec_fd, VIDIOC_QBUF, &buf))
        return errno;
```

Note that it is not allowed to use the Request API for CAPTURE buffers since there are no per-frame settings to report there.

Once the request is fully prepared, it can be queued to the driver:

```
if (ioctl(req_fd, MEDIA_REQUEST_IOC_QUEUE))
        return errno;
```

User-space can then either wait for the request to complete by calling poll() on its file descriptor, or start dequeuing CAPTURE buffers. Most likely, it will want to get CAPTURE buffers as soon as possible and this can be done using a regular :ref:`VIDIOC_DQBUF <VIDIOC_QBUF>`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 172); *backlink*

Unknown interpreted text role "ref".

```
struct v4l2_buffer buf;

memset(&buf, 0, sizeof(buf));
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (ioctl(codec_fd, VIDIOC_DQBUF, &buf))
        return errno;
```

Note that this example assumes for simplicity that for every OUTPUT buffer there will be one CAPTURE buffer, but this does not have to be the case.

We can then, after ensuring that the request is completed via polling the request file descriptor, query control values at the time of its completion via a call to :ref:`VIDIOC_G_EXT_CTRLS <VIDIOC_G_EXT_CTRLS>`. This is particularly useful for volatile controls for which we want to query values as soon as the capture buffer is produced.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 189); *backlink*

Unknown interpreted text role "ref".

```
struct pollfd pfd = { .events = POLLPRI, .fd = req_fd };
poll(&pfd, 1, -1);
...
ctrls.which = V4L2_CTRL_WHICH_REQUEST_VAL;
ctrls.request_fd = req_fd;
if (ioctl(codec_fd, VIDIOC_G_EXT_CTRLS, &ctrls))
        return errno;
```

Once we don't need the request anymore, we can either recycle it for reuse with :ref:`MEDIA_REQUEST_IOC_REINIT`...

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\mediactl\[linux-master][Documentation][userspace-api][media][mediactl]request-api.rst, line 205); *backlink*

Unknown interpreted text role "ref".

```
if (ioctl(req_fd, MEDIA_REQUEST_IOC_REINIT))
        return errno;
```

... or close its file descriptor to completely dispose of it.

```
close(req_fd);
```

Example for a Simple Capture Device

With a simple capture device, requests can be used to specify controls to apply for a given CAPTURE buffer.

```
struct v4l2_buffer buf;
struct v4l2_ext_controls ctrls;
int req_fd;
...
if (ioctl(media_fd, MEDIA_IOC_REQUEST_ALLOC, &req_fd))
        return errno;
...
ctrls.which = V4L2_CTRL_WHICH_REQUEST_VAL;
ctrls.request_fd = req_fd;
if (ioctl(camera_fd, VIDIOC_S_EXT_CTRLS, &ctrls))
        return errno;
```

```
...
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.flags |= V4L2_BUF_FLAG_REQUEST_FD;
buf.request_fd = req_fd;
if (ioctl(camera_fd, VIDIOC_QBUF, &buf))
        return errno;
```

Once the request is fully prepared, it can be queued to the driver:

```
if (ioctl(req_fd, MEDIA_REQUEST_IOC_QUEUE))
        return errno;
```

User-space can then dequeue buffers, wait for the request completion, query controls and recycle the request as in the M2M example above.