Buffers

A buffer contains data exchanged by application and driver using one of the Streaming I/O methods. In the multi-planar API, the data is held in planes, while the buffer structure acts as a container for the planes. Only pointers to buffers (planes) are exchanged, the data itself is not copied. These pointers, together with meta-information like timestamps or field parity, are stored in a struct :c:type:`v4l2_buffer`, argument to the :ref:`VIDIOC_QUERYBUF`, :ref:`VIDIOC_QBUF <VIDIOC_QBUF>` and :ref:`VIDIOC_DQBUF <VIDIOC_QBUF>` ioctl. In the multi-planar API, some plane-specific members of struct :c:type:`v4l2_buffer`, such as pointers and sizes for each plane, are stored in struct :c:type:`v4l2_plane` instead. In that case, struct :c:type:`v4l2_buffer` contains an array of plane structures.

Dequeued video buffers come with timestamps. The driver decides at which part of the frame and with which clock the timestamp is taken. Please see flags in the masks `V4L2_BUF_FLAG_TIMESTAMP_MASK` and `V4L2_BUF_FLAG_TSTAMP_SRC_MASK` in :ref:`buffer-flags`. These flags are always valid and constant across all buffers during the whole video stream. Changes in these flags may take place as a side effect of :ref:`VIDIOC_S_INPUT <VIDIOC_G_INPUT>` or :ref:`VIDIOC_S_OUTPUT <VIDIOC_G_OUTPUT>` however. The `V4L2_BUF_FLAG_TIMESTAMP_COPY` timestamp type which is used by e.g. on mem-to-mem devices is an exception to the rule: the timestamp source flags are copied from the OUTPUT video buffer to the CAPTURE video buffer.

Interactions between formats, controls and buffers

V4L2 exposes parameters that influence the buffer size, or the way data is laid out in the buffer. Those parameters are exposed through both formats and controls. One example of such a control is the `V4L2_CID_ROTATE` control that modifies the direction in which pixels are stored in the buffer, as well as the buffer size when the selected format includes padding at the end of lines.

The set of information needed to interpret the content of a buffer (e.g. the pixel format, the line stride, the tiling orientation or the rotation) is collectively referred to in the rest of this section as the buffer layout.

Controls that can modify the buffer layout shall set the `V4L2_CTRL_FLAG_MODIFY_LAYOUT` flag.

Modifying formats or controls that influence the buffer size or layout require the stream to be stopped. Any attempt at such a modification while the stream is active shall cause the ioctl setting the format or the control to return the `EBUSY` error code. In that case drivers shall also set the `V4L2_CTRL_FLAG_GRABBED` flag when calling :c:func:`VIDIOC_QUERYCTRL` or :c:func:`VIDIOC_QUERY_EXT_CTRL` for such a control while the stream is active.

Note

The :c:func:`VIDIOC_S_SELECTION` ioctl can, depending on the hardware (for instance if the device doesn't include a scaler), modify the format in addition to the selection rectangle. Similarly, the :c:func:`VIDIOC_S_INPUT`, :c:func:`VIDIOC_S_OUTPUT`, :c:func:`VIDIOC_S_STD` and :c:func:`VIDIOC_S_DV_TIMINGS` ioctls can also modify the format and selection rectangles. When those ioctls result in a buffer size or layout change, drivers shall handle that condition as they would handle it in the :c:func:`VIDIOC_S_FMT` ioctl in all cases described in this section.

Controls that only influence the buffer layout can be modified at any time when the stream is stopped. As they don't influence the buffer size, no special handling is needed to synchronize those controls with buffer allocation and the `V4L2_CTRL_FLAG_GRABBED` flag is cleared once the stream is stopped.

Formats and controls that influence the buffer size interact with buffer allocation. The simplest way to handle this is for drivers to always require buffers to be reallocated in order to change those formats or controls. In that case, to perform such changes, userspace applications shall first stop the video stream with the :c:func:`VIDIOC_STREAMOFF` ioctl if it is running and free all buffers with the :c:func:`VIDIOC_REQBUFS` ioctl if they are allocated. After freeing all buffers the `V4L2_CTRL_FLAG_GRABBED` flag for controls is cleared. The format or controls can then be modified, and buffers shall then be reallocated and the stream restarted. A typical ioctl sequence is

1. VIDIOC_STREAMOFF
2. VIDIOC_REQBUFS(0)
3. VIDIOC_S_EXT_CTRLS
4. VIDIOC_S_FMT
5. VIDIOC_REQBUFS(n)
6. VIDIOC_QBUF
7. VIDIOC_STREAMON

The second :c:func:`VIDIOC_REQBUFS` call will take the new format and control value into account to compute the buffer size to allocate. Applications can also retrieve the size by calling the :c:func:`VIDIOC_G_FMT` ioctl if needed.

> **Note**
>
> The API doesn't mandate the above order for control (3.) and format (4.) changes. Format and controls can be set in a different order, or even interleaved, depending on the device and use case. For instance some controls might behave differently for different pixel formats, in which case the format might need to be set first.

When reallocation is required, any attempt to modify format or controls that influences the buffer size while buffers are allocated shall cause the format or control set ioctl to return the `EBUSY` error. Any attempt to queue a buffer too small for the current format or controls shall cause the :c:func:`VIDIOC_QBUF` ioctl to return a `EINVAL` error.

Buffer reallocation is an expensive operation. To avoid that cost, drivers can (and are encouraged to) allow format or controls that influence the buffer size to be changed with buffers allocated. In that case, a typical ioctl sequence to modify format and controls is

1. VIDIOC_STREAMOFF
2. VIDIOC_S_EXT_CTRLS
3. VIDIOC_S_FMT
4. VIDIOC_QBUF
5. VIDIOC_STREAMON

For this sequence to operate correctly, queued buffers need to be large enough for the new format or controls. Drivers shall return a ENOSPC error in response to format change (:c:func:`VIDIOC_S_FMT`) or control changes (:c:func:`VIDIOC_S_CTRL` or :c:func:`VIDIOC_S_EXT_CTRLS`) if buffers too small for the new format are currently queued. As a simplification, drivers are allowed to return a EBUSY error from these ioctls if any buffer is currently queued, without checking the queued buffers sizes.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, **line 127);** *backlink*
>
> Unknown interpreted text role "c:func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, **line 127);** *backlink*
>
> Unknown interpreted text role "c:func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, **line 127);** *backlink*
>
> Unknown interpreted text role "c:func".

Additionally, drivers shall return a EINVAL error from the :c:func:`VIDIOC_QBUF` ioctl if the buffer being queued is too small for the current format or controls. Together, these requirements ensure that queued buffers will always be large enough for the configured format and controls.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, **line 135);** *backlink*
>
> Unknown interpreted text role "c:func".

Userspace applications can query the buffer size required for a given format and controls by first setting the desired control values and then trying the desired format. The :c:func:`VIDIOC_TRY_FMT` ioctl will return the required buffer size.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, **line 140);** *backlink*
>
> Unknown interpreted text role "c:func".

1. VIDIOC_S_EXT_CTRLS(x)
2. VIDIOC_TRY_FMT()
3. VIDIOC_S_EXT_CTRLS(y)
4. VIDIOC_TRY_FMT()

The :c:func:`VIDIOC_CREATE_BUFS` ioctl can then be used to allocate buffers based on the queried sizes (for instance by allocating a set of buffers large enough for all the desired formats and controls, or by allocating separate set of appropriately sized buffers for each use case).

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, **line 150);** *backlink*
>
> Unknown interpreted text role "c:func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, **line 155)**
>
> Unknown directive type "c:type".
>
>   .. c:type:: v4l2_buffer

struct v4l2_buffer

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, line 160)

Unknown directive type "tabularcolumns".

```
.. tabularcolumns:: |p{2.9cm}|p{2.4cm}|p{12.0cm}|
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, line 162)

Unknown directive type "cssclass".

```
.. cssclass:: longtable
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst`, line 164)

Unknown directive type "flat-table".

```
.. flat-table:: struct v4l2_buffer
    :header-rows:  0
    :stub-columns: 0
    :widths:       1 2 10

    * - __u32
      - ``index``
      - Number of the buffer, set by the application except when calling
        :ref:`VIDIOC_DQBUF <VIDIOC_QBUF>`, then it is set by the
        driver. This field can range from zero to the number of buffers
        allocated with the :ref:`VIDIOC_REQBUFS` ioctl
        (struct :c:type:`v4l2_requestbuffers`
        ``count``), plus any buffers allocated with
        :ref:`VIDIOC_CREATE_BUFS` minus one.
    * - __u32
      - ``type``
      - Type of the buffer, same as struct
        :c:type:`v4l2_format` ``type`` or struct
        :c:type:`v4l2_requestbuffers` ``type``, set
        by the application. See :c:type:`v4l2_buf_type`
    * - __u32
      - ``bytesused``
      - The number of bytes occupied by the data in the buffer. It depends
        on the negotiated data format and may change with each buffer for
        compressed variable size data like JPEG images. Drivers must set
        this field when ``type`` refers to a capture stream, applications
        when it refers to an output stream. If the application sets this
        to 0 for an output stream, then ``bytesused`` will be set to the
        size of the buffer (see the ``length`` field of this struct) by
        the driver. For multiplanar formats this field is ignored and the
        ``planes`` pointer is used instead.
    * - __u32
      - ``flags``
      - Flags set by the application or driver, see :ref:`buffer-flags`.
    * - __u32
      - ``field``
      - Indicates the field order of the image in the buffer, see
        :c:type:`v4l2_field`. This field is not used when the buffer
        contains VBI data. Drivers must set it when ``type`` refers to a
        capture stream, applications when it refers to an output stream.
    * - struct timeval
      - ``timestamp``
      - For capture streams this is time when the first data byte was
        captured, as returned by the :c:func:`clock_gettime()` function
        for the relevant clock id; see ``V4L2_BUF_FLAG_TIMESTAMP_*`` in
        :ref:`buffer-flags`. For output streams the driver stores the
        time at which the last data byte was actually sent out in the
        ``timestamp`` field. This permits applications to monitor the
        drift between the video and system clock. For output streams that
        use ``V4L2_BUF_FLAG_TIMESTAMP_COPY`` the application has to fill
        in the timestamp which will be copied by the driver to the capture
        stream.
    * - struct :c:type:`v4l2_timecode`
      - ``timecode``
      - When the ``V4L2_BUF_FLAG_TIMECODE`` flag is set in ``flags``, this
        structure contains a frame timecode. In
        :c:type:`V4L2_FIELD_ALTERNATE <v4l2_field>` mode the top and
        bottom field contain the same timecode. Timecodes are intended to
        help video editing and are typically recorded on video tapes, but
        also embedded in compressed formats like MPEG. This field is
        independent of the ``timestamp`` and ``sequence`` fields.
    * - __u32
      - ``sequence``
      - Set by the driver, counting the frames (not fields!) in sequence.
        This field is set for both input and output devices.
```

```
* - :cspan:`2`

    In :c:type:`V4L2_FIELD_ALTERNATE <v4l2_field>` mode the top and
    bottom field have the same sequence number. The count starts at
    zero and includes dropped or repeated frames. A dropped frame was
    received by an input device but could not be stored due to lack of
    free buffer space. A repeated frame was displayed again by an
    output device because the application did not pass new data in
    time.

    .. note::

        This may count the frames received e.g. over USB, without
        taking into account the frames dropped by the remote hardware due
        to limited compression throughput or bus bandwidth. These devices
        identify by not enumerating any video standards, see
        :ref:`standard`.

* - __u32
  - ``memory``
  - This field must be set by applications and/or drivers in
    accordance with the selected I/O method. See :c:type:`v4l2_memory`
* - union {
  - ``m``
* - __u32
  - ``offset``
  - For the single-planar API and when ``memory`` is
    ``V4L2_MEMORY_MMAP`` this is the offset of the buffer from the
    start of the device memory. The value is returned by the driver
    and apart of serving as parameter to
    :c:func:`mmap()` function not useful for applications.
    See :ref:`mmap` for details
* - unsigned long
  - ``userptr``
  - For the single-planar API and when ``memory`` is
    ``V4L2_MEMORY_USERPTR`` this is a pointer to the buffer (casted to
    unsigned long type) in virtual memory, set by the application. See
    :ref:`userp` for details.
* - struct v4l2_plane
  - ``*planes``
  - When using the multi-planar API, contains a userspace pointer to
    an array of struct :c:type:`v4l2_plane`. The size of
    the array should be put in the ``length`` field of this
    struct :c:type:`v4l2_buffer` structure.
* - int
  - ``fd``
  - For the single-plane API and when ``memory`` is
    ``V4L2_MEMORY_DMABUF`` this is the file descriptor associated with
    a DMABUF buffer.
* - }
  -
* - __u32
  - ``length``
  - Size of the buffer (not the payload) in bytes for the
    single-planar API. This is set by the driver based on the calls to
    :ref:`VIDIOC_REQBUFS` and/or
    :ref:`VIDIOC_CREATE_BUFS`. For the
    multi-planar API the application sets this to the number of
    elements in the ``planes`` array. The driver will fill in the
    actual number of valid elements in that array.
* - __u32
  - ``reserved2``
  - A place holder for future extensions. Drivers and applications
    must set this to 0.
* - __u32
  - ``request_fd``
  - The file descriptor of the request to queue the buffer to. If the flag
    ``V4L2_BUF_FLAG_REQUEST_FD`` is set, then the buffer will be
    queued to this request. If the flag is not set, then this field will
    be ignored.

    The ``V4L2_BUF_FLAG_REQUEST_FD`` flag and this field are only used by
    :ref:`ioctl VIDIOC_QBUF <VIDIOC_QBUF>` and ignored by other ioctls that
    take a :c:type:`v4l2_buffer` as argument.

    Applications should not set ``V4L2_BUF_FLAG_REQUEST_FD`` for any ioctls
    other than :ref:`VIDIOC_QBUF <VIDIOC_QBUF>`.

    If the device does not support requests, then ``EBADR`` will be returned.
    If requests are supported but an invalid request file descriptor is
    given, then ``EINVAL`` will be returned.
```

struct v4l2_plane

enum v4l2_buf_type

- 14
- Buffer for metadata output, see :ref:`metadata`.

Buffer Flags

- ``V4L2_BUF_FLAG_KEYFRAME``
    - 0x00000008
    - Drivers set or clear this flag when calling the ``VIDIOC_DQBUF``
      ioctl. It may be set by video capture devices when the buffer
      contains a compressed image which is a key frame (or field), i. e.
      can be decompressed on its own. Also known as an I-frame.
      Applications can set this bit when ``type`` refers to an output
      stream.
  * .. _`V4L2-BUF-FLAG-PFRAME`:

    - ``V4L2_BUF_FLAG_PFRAME``
    - 0x00000010
    - Similar to ``V4L2_BUF_FLAG_KEYFRAME`` this flags predicted frames
      or fields which contain only differences to a previous key frame.
      Applications can set this bit when ``type`` refers to an output
      stream.
  * .. _`V4L2-BUF-FLAG-BFRAME`:

    - ``V4L2_BUF_FLAG_BFRAME``
    - 0x00000020
    - Similar to ``V4L2_BUF_FLAG_KEYFRAME`` this flags a bi-directional
      predicted frame or field which contains only the differences
      between the current frame and both the preceding and following key
      frames to specify its content. Applications can set this bit when
      ``type`` refers to an output stream.
  * .. _`V4L2-BUF-FLAG-TIMECODE`:

    - ``V4L2_BUF_FLAG_TIMECODE``
    - 0x00000100
    - The ``timecode`` field is valid. Drivers set or clear this flag
      when the ``VIDIOC_DQBUF`` ioctl is called. Applications can set
      this bit and the corresponding ``timecode`` structure when
      ``type`` refers to an output stream.
  * .. _`V4L2-BUF-FLAG-PREPARED`:

    - ``V4L2_BUF_FLAG_PREPARED``
    - 0x00000400
    - The buffer has been prepared for I/O and can be queued by the
      application. Drivers set or clear this flag when the
      :ref:`VIDIOC_QUERYBUF`,
      :ref:`VIDIOC_PREPARE_BUF <VIDIOC_QBUF>`,
      :ref:`VIDIOC_QBUF` or
      :ref:`VIDIOC_DQBUF <VIDIOC_QBUF>` ioctl is called.
  * .. _`V4L2-BUF-FLAG-NO-CACHE-INVALIDATE`:

    - ``V4L2_BUF_FLAG_NO_CACHE_INVALIDATE``
    - 0x00000800
    - Caches do not have to be invalidated for this buffer. Typically
      applications shall use this flag if the data captured in the
      buffer is not going to be touched by the CPU, instead the buffer
      will, probably, be passed on to a DMA-capable hardware unit for
      further processing or output. This flag is ignored unless the
      queue is used for :ref:`memory mapping <mmap>` streaming I/O and
      reports :ref:`V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS
      <V4L2-BUF-CAP-SUPPORTS-MMAP-CACHE-HINTS>` capability.
  * .. _`V4L2-BUF-FLAG-NO-CACHE-CLEAN`:

    - ``V4L2_BUF_FLAG_NO_CACHE_CLEAN``
    - 0x00001000
    - Caches do not have to be cleaned for this buffer. Typically
      applications shall use this flag for output buffers if the data in
      this buffer has not been created by the CPU but by some
      DMA-capable unit, in which case caches have not been used. This flag
      is ignored unless the queue is used for :ref:`memory mapping <mmap>`
      streaming I/O and reports :ref:`V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS
      <V4L2-BUF-CAP-SUPPORTS-MMAP-CACHE-HINTS>` capability.
  * .. _`V4L2-BUF-FLAG-M2M-HOLD-CAPTURE-BUF`:

    - ``V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF``
    - 0x00000200
    - Only valid if struct :c:type:`v4l2_requestbuffers` flag ``V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_
      set. It is typically used with stateless decoders where multiple
      output buffers each decode to a slice of the decoded frame.
      Applications can set this flag when queueing the output buffer
      to prevent the driver from dequeueing the capture buffer after
      the output buffer has been decoded (i.e. the capture buffer is
      'held'). If the timestamp of this output buffer differs from that
      of the previous output buffer, then that indicates the start of a
      new frame and the previously held capture buffer is dequeued.
  * .. _`V4L2-BUF-FLAG-LAST`:

    - ``V4L2_BUF_FLAG_LAST``
    - 0x00100000
    - Last buffer produced by the hardware. mem2mem codec drivers set
      this flag on the capture queue for the last buffer when the
      :ref:`VIDIOC_QUERYBUF` or
      :ref:`VIDIOC_DQBUF <VIDIOC_QBUF>` ioctl is called. Due to
      hardware limitations, the last buffer may be empty. In this case
      the driver will set the ``bytesused`` field to 0, regardless of
      the format. Any subsequent call to the
      :ref:`VIDIOC_DQBUF <VIDIOC_QBUF>` ioctl will not block anymore,
      but return an ``EPIPE`` error code.
  * .. _`V4L2-BUF-FLAG-REQUEST-FD`:

```
   - ``V4L2_BUF_FLAG_REQUEST_FD``
   - 0x00800000
   - The ``request_fd`` field contains a valid file descriptor.
 * .. _`V4L2-BUF-FLAG-TIMESTAMP-MASK`:

   - ``V4L2_BUF_FLAG_TIMESTAMP_MASK``
   - 0x0000e000
   - Mask for timestamp types below. To test the timestamp type, mask
     out bits not belonging to timestamp type by performing a logical
     and operation with buffer flags and timestamp mask.
 * .. _`V4L2-BUF-FLAG-TIMESTAMP-UNKNOWN`:

   - ``V4L2_BUF_FLAG_TIMESTAMP_UNKNOWN``
   - 0x00000000
   - Unknown timestamp type. This type is used by drivers before Linux
     3.9 and may be either monotonic (see below) or realtime (wall
     clock). Monotonic clock has been favoured in embedded systems
     whereas most of the drivers use the realtime clock. Either kinds
     of timestamps are available in user space via
     :c:func:`clock_gettime` using clock IDs ``CLOCK_MONOTONIC``
     and ``CLOCK_REALTIME``, respectively.
 * .. _`V4L2-BUF-FLAG-TIMESTAMP-MONOTONIC`:

   - ``V4L2_BUF_FLAG_TIMESTAMP_MONOTONIC``
   - 0x00002000
   - The buffer timestamp has been taken from the ``CLOCK_MONOTONIC``
     clock. To access the same clock outside V4L2, use
     :c:func:`clock_gettime`.
 * .. _`V4L2-BUF-FLAG-TIMESTAMP-COPY`:

   - ``V4L2_BUF_FLAG_TIMESTAMP_COPY``
   - 0x00004000
   - The CAPTURE buffer timestamp has been taken from the corresponding
     OUTPUT buffer. This flag applies only to mem2mem devices.
 * .. _`V4L2-BUF-FLAG-TSTAMP-SRC-MASK`:

   - ``V4L2_BUF_FLAG_TSTAMP_SRC_MASK``
   - 0x00070000
   - Mask for timestamp sources below. The timestamp source defines the
     point of time the timestamp is taken in relation to the frame.
     Logical 'and' operation between the ``flags`` field and
     ``V4L2_BUF_FLAG_TSTAMP_SRC_MASK`` produces the value of the
     timestamp source. Applications must set the timestamp source when
     ``type`` refers to an output stream and
     ``V4L2_BUF_FLAG_TIMESTAMP_COPY`` is set.
 * .. _`V4L2-BUF-FLAG-TSTAMP-SRC-EOF`:

   - ``V4L2_BUF_FLAG_TSTAMP_SRC_EOF``
   - 0x00000000
   - End Of Frame. The buffer timestamp has been taken when the last
     pixel of the frame has been received or the last pixel of the
     frame has been transmitted. In practice, software generated
     timestamps will typically be read from the clock a small amount of
     time after the last pixel has been received or transmitten,
     depending on the system and other activity in it.
 * .. _`V4L2-BUF-FLAG-TSTAMP-SRC-SOE`:

   - ``V4L2_BUF_FLAG_TSTAMP_SRC_SOE``
   - 0x00010000
   - Start Of Exposure. The buffer timestamp has been taken when the
     exposure of the frame has begun. This is only valid for the
     ``V4L2_BUF_TYPE_VIDEO_CAPTURE`` buffer type.
```

enum v4l2_memory

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst, line 682)

Unknown directive type "tabularcolumns".

```
.. tabularcolumns:: |p{5.0cm}|p{0.8cm}|p{11.5cm}|
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\media\v4l\(linux-master)(Documentation)(userspace-api)(media)(v4l)buffer.rst, line 684)

Unknown directive type "flat-table".

```
.. flat-table::
   :header-rows:  0
   :stub-columns: 0
   :widths:       3 1 4

   * - ``V4L2_MEMORY_MMAP``
     - 1
     - The buffer is used for :ref:`memory mapping <mmap>` I/O.
```

```
    * - ``V4L2_MEMORY_USERPTR``
      - 2
      - The buffer is used for :ref:`user pointer <userp>` I/O.
    * - ``V4L2_MEMORY_OVERLAY``
      - 3
      - [to do]
    * - ``V4L2_MEMORY_DMABUF``
      - 4
      - The buffer is used for :ref:`DMA shared buffer <dmabuf>` I/O.
```

Memory Consistency Flags

Timecodes

The :c:type:`v4l2_buffer_timecode` structure is designed to hold a :ref:`smpte12m` or similar timecode. (struct :c:type:`timeval` timestamps are stored in the struct :c:type:`v4l2_buffer` timestamp field.)

struct v4l2_timecode

Timecode Types

```
        -
 * - ``V4L2_TC_TYPE_60FPS``
   - 5
   -
```

Timecode Flags